Trusted Mobile Platform
Protocol Specification Document

04/05/2004

Trusted Mobile Platform
NTT DoCoMo, IBM, Intel Corporation

File Name: tmp_protocol_spec_rev1_00__20040405.doc

Trusted Mobile Platform Protocol Specification

Rev. 1.00

04/05/2004

Authors

Selim Aissi (Intel)

Hiroshi Maruyama (IBM)

Fumiaki Miura (NTT DoCoMo)

Taiga Nakamura (IBM)

Daniel Saito (NTT DoCoMo)

Atsushi Takeshita (NTT DoCoMo)

Dave Wheeler (Intel)

Sachiko Yoshihama (IBM)

Copyright Notice

Status

This is a stable revision of the Trusted Mobile Platform Protocol Specification that was agreed upon by Trusted Mobile Platform promoters.

Table of Contents

1 Introduction

The Trusted Mobile Platform is a project to establish technologies for enhancing platform security and trustworthiness on wireless devices. It is based on a platform architecture comprised of hardware and software with security features that can be used as a basis for establishing trust in the entire platform. With the enhanced trust, it enables supporting various mission-critical business applications, such as those described in the Phase Zero document **[PhaseZero]**, and enables data service businesses to be brought to the mobile industry.

Security is a significant concern for mobile device users. How can data exchanged over the air be protected from eavesdropping? How can the user be certain that information received from a service is authentic and has not been changed since it was created? How is the device protected from malicious downloaded programs such as viruses? The Trusted Mobile Platform specifications provide appropriate protections which address a multiplicity of security concerns, including the above ones.

This document defines a set of protocols that enable the Trusted Mobile Platform, described in the *Hardware Architecture Description* **[HWAD]** and *Software Architecture Description* **[SWAD]** documents, to communicate with other platforms securely. The intent of the Protocol specification document is not to create a complete set of protocols from scratch. Rather, this document investigates available and emerging open standards, by identifying specific areas that are currently missing to support trusted mobile devices.

1.1 Structure of This Document

This document is organized as follows. Section 2 revisits the general principles described in the Phase Zero document **[PhaseZero]** and the multi-layered approach to the protocol design. Section 3 explains how the protocols defined in this document can be used in typical scenarios. Section 4 defines a profile of WS-Security standards **[WS-Security]** for the Trusted Mobile devices. Protocols specifically designed to support TCG functionality **[TCG]** is defined in Section 5. One of the major threats identified in the Phase Zero document **[PhaseZero]** is a loss of communication during a transaction. The Fair Contract Signing Protocol defined in Section 6 addresses this threat. Protocols supporting for device management, including key management, are specified in Section 7. Section 8 demonstrates a generic model of access control among various control domains. . Section 9 addresses various security protocol standards and how they relate to this specification. The document concludes with numerous security considerations when applying these protocols described in Section 10.

1.2 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 **[Keywords]**

2 Protocol Design General Principles

In this section the general principles used to define the Trusted Mobile Platform protocols and the scope and design framework are described.

2.1 Trusted Mobile Platform Design Principles

The following design principles were identified in the Phase Zero document **[PhaseZero]**.

Enhanced Security: The Trusted Mobile Platform specifications shall provide a set of technologies to offer a range of security applications to enhance platform security on wireless devices.

Deployment Target: The Trusted Mobile Platform specifications shall target deployments in 2005 or later.

Leverage Existing Standards: The Trusted Mobile Platform specifications shall use existing standards wherever possible, unless they can explicitly justify the need for developing a new specification.

Friendliness with the Internet: *We are living in the Eco-systems*. The Trusted Mobile Platform specifications shall be friendly with the Internet standards and technologies.

Win-win Technology: The Trusted Mobile Platform specifications shall support applications that enable win-win situations to everyone in the value chain, minimize conflict with existing players and barrier to entry, and leverage existing player.

Security Concerns: The Trusted Mobile Platform specifications shall explicitly describe any security concern if there exists such a threat that they are not able to address.

User Experience: The Trusted Mobile Platform specifications shall enable user experience to be very simple and intuitive, so that they accommodate consumer applications as well as business applications.

Efficiency: The Trusted Mobile Platform specifications shall enable efficient implementations on mobile devices with limited resources, such as CPU, memory, IO, and efficient use of communication bandwidth.

Security Levels: The Trusted Mobile Platform specifications shall provide multiple security levels to accommodate a variety of security requirements or trade-offs from various applications.

Security Analysis: We shall consider the possibility of applying security analysis techniques, either formal ones or methodological ones, to the Trusted Mobile Platform specifications and implementations.

Fail-Secure: The Trusted Mobile Platform specifications shall be designed with possible failures of any constituents taken into account. These failures shall not introduce new vulnerabilities.

Privacy: Privacy shall be considered wherever applicable.

Deployment and Maintenance: The Trusted Mobile Platform specifications shall support application systems that are easy to deploy and maintain.

Simplified Model: The Trusted Mobile Platform specifications shall reduce the number of players and simplify interdependency between them, as long as they are able to achieve the required security levels, in order to facilitate and accelerate adoption of the platform.

Simple Specifications: The Trusted Mobile Platform specifications shall be simple and concise.

Scalability: The Trusted Mobile Platform specifications shall support application systems that are easy to scale to accommodate millions of users in a relatively short period of time.

Extensibility: The Trusted Mobile Platform specifications shall be extensible so that it is easy to add new functionalities and mechanisms to address new types of applications and threats.

Mobility: The Trusted Mobile Platform specifications shall address inherent characteristics of mobile communications. Compared with wired communications, mobile communications suffers from a higher probability of being disconnected in the middle of transactions, and no recovery for a long time or practically forever.

Transaction Time: Minimizing transaction time is one of the most important performance elements for better user experiences. The Trusted Mobile Platform specifications shall enable optimized transaction time for every application.

Operator Domain Trust: The operator domain is typically considered to be safer than that of the Internet. Due to the introduction of open terminals and hybrid network architectures combining various short range private communications that may be connected to the Internet, this assumption tends to be weaker. The Trusted Mobile Platform specifications shall address the aspect of reducing operator domain trust issues.

2.2 Layered Protocol

Based on these design principles, the Trusted Mobile Platform protocol has the layered architecture as shown in *Figure 2-1 Trusted Mobile Platform Protocol Layered Architecture*.

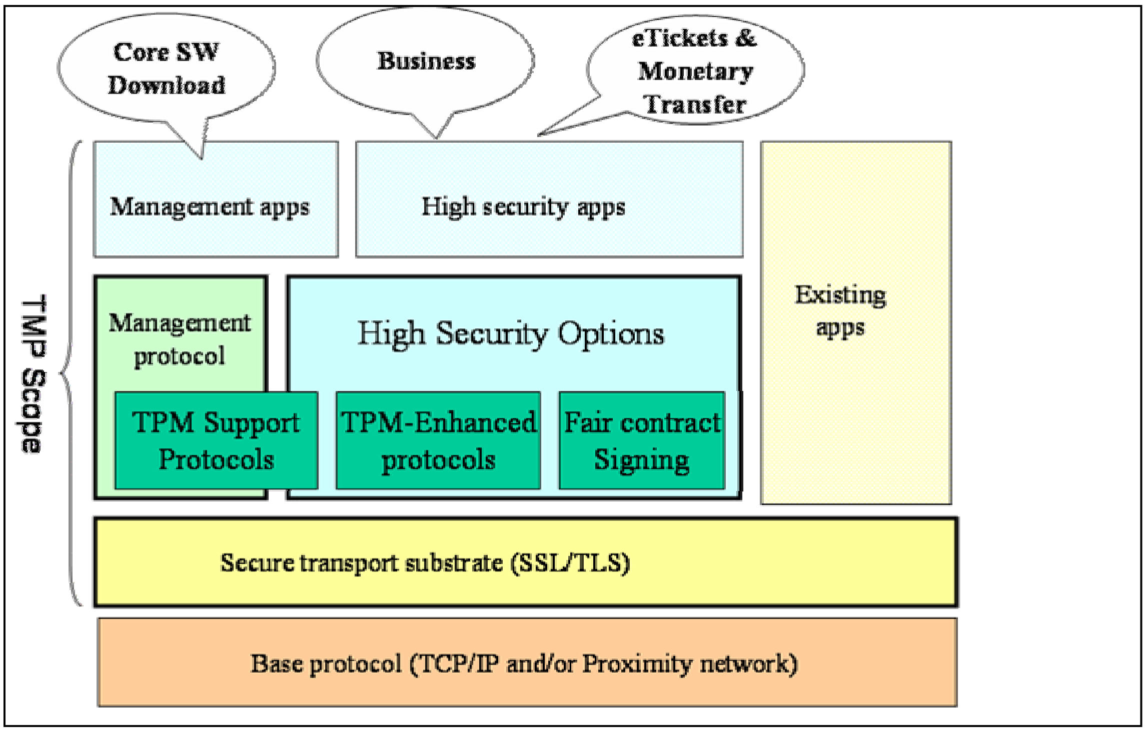


Figure 2-1Trusted Mobile Platform Protocol Layered Architecture

At the bottom, Trusted Mobile Platform assumes the base protocol, which consists of TCP/IP **[TCP/IP]** and some form of proximity connection, such as Bluetooth **[Bluetooth]** or ISO 14443 **[ISO14443]**. These base protocols may support some form of security such as IPSec **[IPSec]**, The Trusted Mobile Platform protocol does not assume any of them. Non-protected applications, such as normal Web browsing, can use the base protocol without any further protection.

The next layer of the Trusted Mobile Platform protocol is the secure transport substrate, which is typically the transport layer Security (TLS, also known as Secure Sockets Layer or SSL) **[TLS]**. SSL/TLS is the most widely used security protocol on today's Internet. All Trusted Mobile devices are RECOMMENDED to support SSL/TLS with the current Internet practices. These practices include, embedding a set of trusted root CA certificates (trust anchors) in the applications such as Web browsers. SSL/TLS mandates the server side authentication based on server's X.509 certificates, but the client side authentication is optional. In Trusted Mobile Devices, client authentication should be done either through traditional password-based authentication (either by HTTP basic authentication **[HTTPBasicAuth]** or HTML form-based authentication **[HTML]**) or by SSL/TLS client authentication [TLS]. Supporting SSL/TLS allows the users of Trusted Mobile devices access the large number of the Internet applications (e.g., Web sites) currently available. Many Internet applications, such as secure Web browsing (https) and secure mail box (pops) can use this layer.

Trusted Mobile devices may optionally have a set of protocols specific to Trusted Mobile devices, called Trusted Mobile Platform High Security Options. These options provide higher level of protections such as end-to-end protection by taking advantage of the Trusted Mobile device, most notably the use of the TPM (Trusted Platform Module). This layer also defines a protocol that specifically addresses the concern of unstable wireless network when conducting transactions. This is done through the Fair Contract Signing Protocol. Applications that require higher level of protection, such as e-Ticket and B2B/B2E, may use the protocols defined in this layer, with or without the transport layer protection provided by SSL/TLS.

The device also has a set of management protocols that support management of the device. This includes key management, software download, and general device management. The core software download scenario described in the Phase Zero document **[PhaseZero]** may use this layer of the Trusted Mobile Platform protocol.

2.3 Summary of Trusted Mobile Platform Protocol Requirements and Solutions

The following table summarizes the identified requirements and corresponding Trusted Mobile Platform solutions.

Requirement	**Possible Trusted Mobile Platform solutions**
End-to-end confidentiality/Integrity	SSL/TLS, Profiled WS-Security
Mutual authentication	SSL/TLS (+HTTP Auth), Profiled WS-Security
Digital signature / Non-repudiation	XML Signature (profiled), Contract (new)
Digital ticket format (authorization format)	SAML
Prevention of ticket duplication	Remote data binding protocol (new)
Authorization protocol	WS-Security+WS-Trust (profiled)
Robustness against network failure & un-trusted party	Optimistic fair contract signing protocol (new)
Key management infrastructure	WS-Trust, X.509
Access control language	XACML
Device integrity	TCG-defined X.509 attribute certificate, RSA_PCR algorithm (new)
Device integrity infrastructure	TCG credentials + WS-Trust (profiled)
Device properties	TBD
Device management	TBD

3 Protocols Overview

This section provides an overview of the protocols defined for the Trusted Mobile Platform. This section is informative, and is not intended to replace any normative specifications, but to provide the reader with descriptive and graphical representation of how the Trusted Mobile Platform protocols are utilized to support application scenarios. Should there be a conflict between this section and other normative specifications, the normative specifications shall take precedence.

The second section of this document provides a high level view of the use of Trusted Mobile Platform protocols, followed by examples of their use through the Phase Zero scenarios **[PhaseZero]**. The purpose of the Trusted Mobile Platform Phase Zero Scenarios section is to provide traceability to the Phase Zero requirements for protocols. The scenario examples are informative.

3.1 Actors

The following sections present several scenarios with different entities such as organizations, people, or physical objects that are assumed to participate in the scenarios. Some terms were imported from **[TCG]** while others are defined in this section for convenience. For flexibility, different terms are used to describe each entity that act out a particular role. In the real-world, a single entity may act multiple roles. For example, in the typical consumer market, the device user is usually the device owner.

TPM	The Trusted Platform Module as defined in **[TCG]**.
TMD	A Trusted Mobile Device (TMD) is a device that implements the Trusted Mobile Platform specifications.
Device Manufacturer	The manufacturer that produces TMD.
Software Configuration Service	A service that configures the core software of a TMD. The software configuration service is presumably trusted by the device manufacture and the VE, and is responsible for keeping track of software configuration that is known to be trusted. The software configuration service registers the integrity metrics of the known configurations with the validation entity.
VE	A Validation Entity (VE) is a trusted third party vouches for a TMD that it is working properly and in an expected state.
Privacy CA	A trusted third party that issues platform identities.
Service Provider	Provides application services some of which require the trust mechanism provided by Trusted Mobile Platform specifications. E.g., the merchant who issues event tickets in the e-Ticket scenario.
Device Owner	The owner of a TMD.
Device User	The user of a TMD.

3.2 Trusted Mobile Platform Phase Zero Scenarios

The following sections map the Trusted Mobile Platform protocols to the Phase Zero scenarios.

3.2.1 Issuance of an Identity and Subscription to a Service

Figure 3-1 shows a typical sequence of actions that occur when a platform identity is issued and the user subscribes to a service using that identity. This particular scenario illustrates a case in which the user subscribes to a service immediately after the identity is issued. However, it is also possible that a user subscribes to multiple services using a single identity

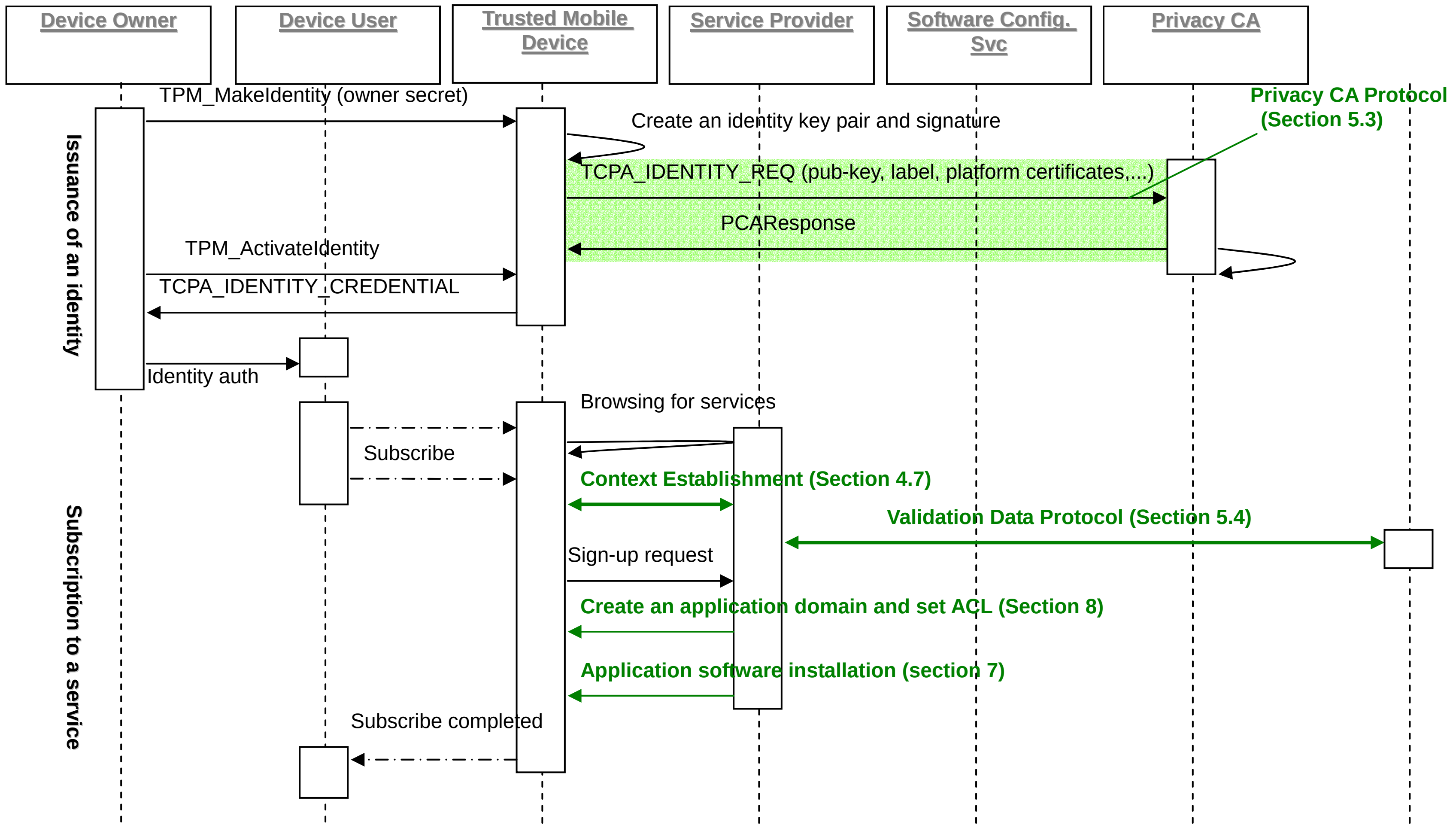

Device Owner
Device User
Trusted Mobile Device
Service Provider
Software Config. Svc
Privacy CA
Privacy CA Protocol (Section 5.3)
Issuance of an identity
TPM_MakeIdentity (owner secret)
Create an identity key pair and signature
TCPA_IDENTITY_REQ (pub-key, label, platform certificates,...)
PCAResponse
TPM_ActivateIdentity
TCPA_IDENTITY_CREDENTIAL
Identity auth
Subscription to a service
Browsing for services
Subscribe
Context Establishment (Section 4.7)
Validation Data Protocol (Section 5.4)
Sign-up request
Create an application domain and set ACL (Section 8)
Application software installation (section 7)
Subscribe completed

Figure 3-1 Issuance of an Identity and Subscription to a Service

The issuance of an identity may take place in the following sequence:

1) The device owner requests to create a platform identity through the user interface of the TMD.
2) The TMD calls `TPM_MakeIdentity` to generate a new asymmetric key-pair and the identity-binding signature that links the information necessary to request an identity, such as the created key-pair, an identity name, endorsement certificate, platform certificate, etc. The device subsequently calls `TSS_CollateIdentityRequest` to assemble all the information.
3) The assembled information is sent to the privacy CA, which then creates and returns the identity certificate. The protocol for exchanging identity request and response is defined as the Privacy CA protocol in section 5.3.
4) The TMD decrypts the data, check if it is intended for the TPM the device contains, and recovers the identity certificate. These processes are done using `TPM_ActivateIdentity` and `TSS_RecoverIdentity` calls as defined in **[TCG]**.
5) An identity authorization secret may be passed to the device user for later use. The identity authorization allows the device user to use the platform identity on behalf of the device owner.

The subscription to a service may take place in the following sequence:

6) The device user browses services provided by a service provider. When he/she decides to subscribe, the TMD creates a secure session using the Context Establishment Protocol (Section 4.7).
7) The integrity metrics of the TMD, which is sent to the service provider as a part of the Context Establishment Protocol, is sent to the VE using the Validation Data Protocol (See section 5.4). The VE checks the integrity metrics against its registry, and responds with the trust status of the metrics. For example, if the integrity metrics matches the one that is already registered by the Software Configuration Service as a “safe” configuration, the VE vouches for it. If the integrity metrics is unknown, or known to be untrusted, the VE will return a warning.
8) The TMD send subscription request to the service provider. If the service provider is satisfied by the validation result from the VE, it creates an application domain and properly configures the access control list for the domain (See section 8).
9) The service provider then install the application software into the newly created application domain using the Terminal Management capability described in section 7.

Note that there may be many alternatives to implement this scenario. For example:

- The identity authorization may be passed to the service provider instead of the device user. This may be a useful scenario if the device owner trusts a service provider and allows the user to subscribe to any service provided by that particular service provider. For example, in an enterprise scenario, the device owner (i.e., IT division of the company) allows the device user (i.e., employee) to subscribe to any service on the intranet (i.e., the services provided by the company).
- The Context Establishment protocol may be replaced by the enhanced SSL/TLS defined in section 5.5, since it also provides a way to exchange the integrity metrics and to setup secure session.
- The figure illustrates a scenario in which the pull mode of the Validation Data Protocol is used. Alternatively, the push mode of the Validation Data Protocol may also be used, in which case the TMD obtains the validation certificate from the VE in advance, and send it along with the subscription request message.

The Privacy-CA is concerned with the following security problems:

Issue/Concern	Mitigation Technique
A non-owner requests an identity	The action of creating an identity is protected by the owner authorization secret
A non-genuine TPM requests an identity	The privacy CA verifies endorsement and platform credential sent from the TMD

The Device Owner is concerned with the following security problems:

Issue/Concern	Mitigation Technique
The identity request is peeked/stolen by an eavesdropper	The request is symmetrically encrypted by using a random nonce as a key, which is asymmetrically encrypted by the public key of the privacy CA, thus the request can be decrypted only the requested privacy CA.
The issued identity is peeked/stolen by an eavesdropper	The issued identity is asymmetrically encrypted by using the requesting TPM's public endorsement key, thus it can be decrypted only by the requesting TPM, since the private endorsement key never leaves the TPM.
A rogue pretends to be the privacy CA	Since the request is encrypted by the public key of the privacy CA, only the genuine privacy CA, that the owner intended to request, can decrypt it.
A malicious user tries to subscribe to a service using an identity, which he/she is not authorized to use	Only the user, who is authorized by the device owner to use an identity, can present an identity authorization secret.
A rogue creates an application domain	The admin authorization secret has to be given to the service provider..

The Service Provider is concerned with the following security problems:

Issue/Concern	Mitigation Technique
A non-genuine TMD subscribes to a service	The attestation signature, sent in the Context Establishment Protocol, proves that the requester owns an identity certified by a trusted privacy CA. The privacy CA knows that the identity is associated with a particular TPM and a TMD.
The software on the TMD is compromised	The integrity metrics in the attestation signature (sent in the Context Establishment Protocol) proves that the platform software is in an expected state, which is verified against the VE.
Eavesdropper peeks the messages between the device and the service provider	Each message in a secure context, established by the Context Establishment Protocol, is protected by the session secret.
Replay attack	Each message in a secure context, established by the Context Establishment Protocol, has a unique sequence id, thus the reply attack can be detected.
Installed application software/data is unexpectedly modified	The application domain belongs to the service provider and is protected even from the device owner.

3.2.2 E-Ticket Scenario Overview

The e-Ticket scenario defines the use of mobile devices to purchase and utilize tickets for access to physical events (e.g. sporting events, concerts, movies, etc.) The steps required to enact the e-Ticket scenario are outlined in **Figure 3-2**, on the next page.

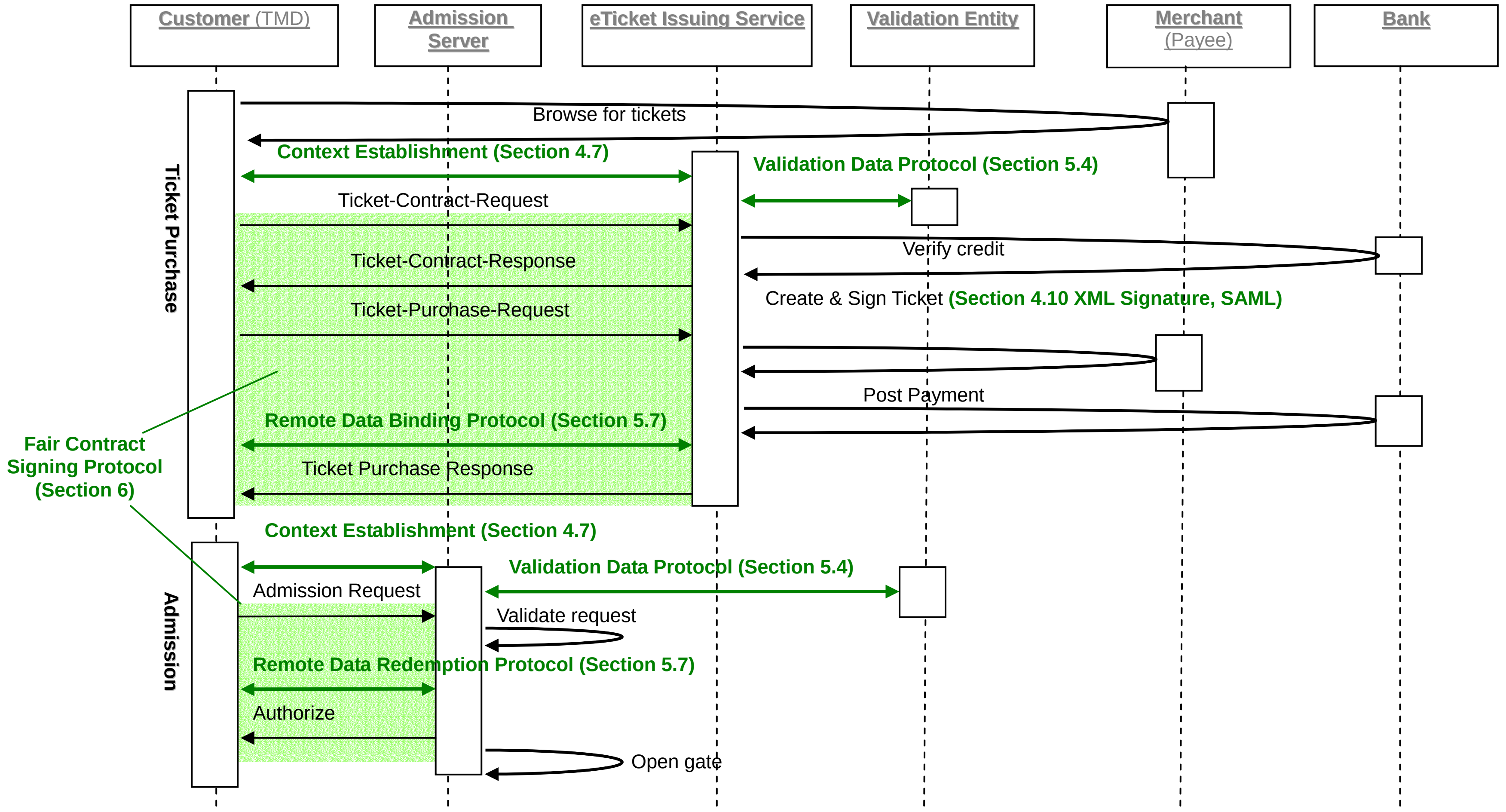

Customer (TMD)
Admission Server
eTicket Issuing Service
Validation Entity
Merchant (Payee)
Bank
Browse for tickets
Context Establishment (Section 4.7)
Validation Data Protocol (Section 5.4)
Ticket-Contract-Request
Verify credit
Ticket-Contract-Response
Create & Sign Ticket (Section 4.10 XML Signature, SAML)
Ticket-Purchase-Request
Post Payment
Remote Data Binding Protocol (Section 5.7)
Ticket Purchase Response
Ticket Purchase
Fair Contract Signing Protocol (Section 6)
Context Establishment (Section 4.7)
Validation Data Protocol (Section 5.4)
Admission Request
Validate request
Remote Data Redemption Protocol (Section 5.7)
Authorize
Open gate
Admission

Figure 3-2 E-Ticket Scenario

1) First, the device user browses tickets of the event that he/she is interested in.
2) The user requests ticket purchase through the user interface. The TMD obtains integrity metrics from TPM (i.e., by calling *TPM_Quote* with passing identity authorization secret to TPM). The TMD then sends integrity metrics to the e-Ticket issuing service (i.e., Service Provider) using the Context Establishment Protocol (Section 4.7). Similar to the previous scenario, the Context Establishment Protocol may replace the enhanced SSL/TLS.
3) The e-Ticket issuing service then asks the VE to validate the integrity metrics.
4) When the e-Ticket issuing service is satisfied by the validation result, the e-Ticket issuing service and the TMD can exchange a purchase contract by the Fair Contract Signing Protocol (Section 6).
5) As the contract progresses, the e-Ticket issuing service may communicate with a bank to verify if the user has enough credit to buy the tickets.
6) The electrical representation of tickets is created by the merchant (or the event promoter) in the form of a SAML assertion, and signed by the merchant to avoid forgery. The e-Ticket issuing service installs the ticket into the TMD by the Remote Data Binding Protocol (Section 5.7), ensuring that the tickets can be accessed only by the ticket issuing service.

Upon admission to the event venue, e-Tickets may be exchanged in the following sequence:

7) When arriving at the venue, the device communicates with the Admission Server, possibly through the proximity protocol such as Bluetooth. The TMD again obtains the integrity metrics from TPM by using the identity authorization secret, and sets up a secure session with the Admission Server using the Context Establishment Protocol.
8) The Admission Server then asks the VE to validate the integrity metrics.
9) When the Admission Server is satisfied by the validation result, the Admission Server and the device exchange the Admission request and response in the Fair Contract Signing Protocol.
10) The Admission Server redeems the e-Ticket by the Remote Data Redemption Protocol.
11) Upon completion of ticket exchange, the Admission Server opens the gate to let the user in.

Note that similar to the previous section, there are several alternatives to implement this scenario; for example, enhanced SSL/TLS handshake can replace the Context Establishment Protocol.

3.2.3 An Example Ticket Purchasing Protocol

Note that the Ticket Purchasing Protocol described in this section is an example, and this document does not define an application protocol.

The **Ticket Purchasing Protocol (TPP)** is initiated by the user of a device to request the purchase of a particular ticket at a particular price. Attributes of the ticket may vary, and are determined by browsing for the appropriate ticket to purchase [step 1]. Therefore the ticket attributes are an input to the TTP. These attributes shall include, but not be limited to, the following elements:

- Event name
- Ticket Event Date (the date or dates or dates range the ticket is used for)
- Ticket Issuer
- Ticket Serial Number
- Ticket Purchase Timestamp
- Ticket Type (e.g. child, adult, student, senior, etc)
- Ticket Usage (e.g. one-time-use, all-day-use, n-times-use, etc)
- Ticket Price (Cost), including a breakdown to identify discounts, taxes, handling fees, etc

Multiple tickets are handled by having a sequence of separate ticket attributes grouped together with a count for the number of tickets and a total price for all the grouped tickets. The TTP transmits the ticket attributes in a Ticket-Purchase-Request to the ticket issuer. The Ticket-Purchase-Request message shall be protected from unauthorized disclosure (privacy) and unauthorized modification (integrity).

The ticket issuer is concerned with the following security problems:

Issue/Concern	Mitigation Technique
Alteration of the purchased ticket details after ticket issuance	Ticket is digitally signed by issuer
Forgery of a ticket	
Duplication of purchased tickets	The Remote Data Binding Protocol ensures that the tickets are encrypted by a key that is known only to the ticket issuer, thus event the device owner cannot access the tickets data.
A ticket is used multiple times	The Remote Data Redemption Protocol ensures that the ticket is removed from the TMD after it is used.
Issuing a ticket that is not paid for (unfairness)	TPP is implemented using a fair-exchange protocol

The Customer (ticket buyer) is concerned with the following security problems:

Issue/Concern	Mitigation Technique
Hijacking a purchased ticket over the air	Ticket is delivered over a secure tunnel
Stealing a ticket at the ticket issuer before the ticket is sent	Ticket is linked to user's handheld device by digital signature
Stealing a ticket from the user's handheld device	Ticket is transmitted only to trusted platforms that protect ticket
Purchasing a ticket from an unauthorized ticket issuer (buying a forgery)	Ticket issuer is authenticated before purchase details are transmitted
Being charged for a ticket that is never delivered (unfairness)	TPP is implemented using a fair-exchange protocol

3.2.4 Mobile Wallet Scenario

In the Mobile Wallet scenario, we use a Trusted Mobile Device to purchase prepaid value tokens from the value token brokerage via a value token charger. The TMD can then purchase in-expensive items, such as a can of soda, from vending machine using value tokens. Figure 3-3 shows an example sequence that may occur in this scenario.

Note: This document does not intend to specify an e-cash or micro payment protocol but rather to show an example of how technologies defined in this specification can be used to build such an application.

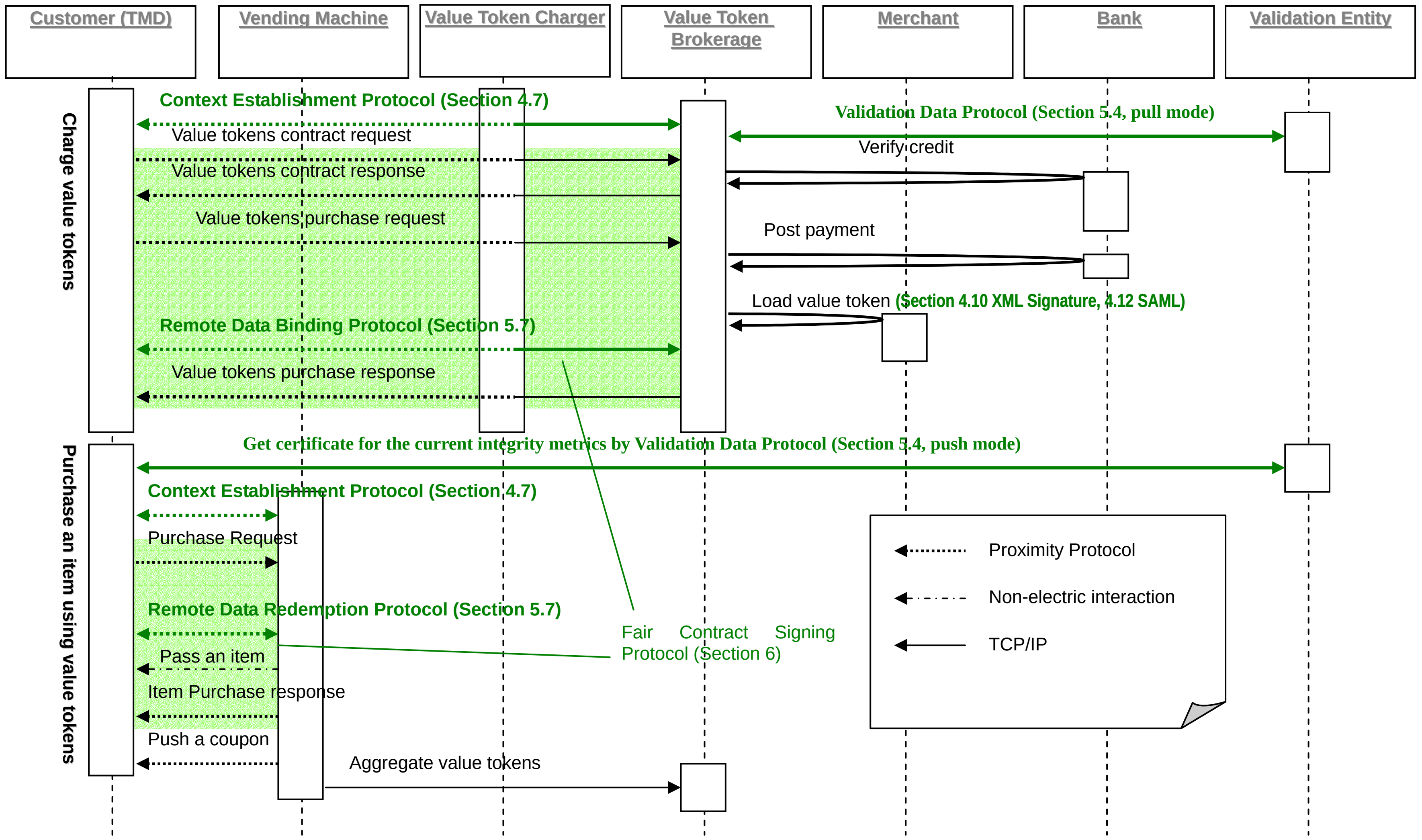

Customer (TMD)
Vending Machine
Value Token Charger
Value Token Brokerage
Merchant
Bank
Validation Entity
Charge value tokens
Context Establishment Protocol (Section 4.7)
Validation Data Protocol (Section 5.4, pull mode)
Value tokens contract request
Verify credit
Value tokens contract response
Value tokens purchase request
Post payment
Load value token (Section 4.10 XML Signature, 4.12 SAML)
Remote Data Binding Protocol (Section 5.7)
Value tokens purchase response
Purchase an item using value tokens
Get certificate for the current integrity metrics by Validation Data Protocol (Section 5.4, push mode)
Context Establishment Protocol (Section 4.7)
Purchase Request
Remote Data Redemption Protocol (Section 5.7)
Fair Contract Signing Protocol (Section 6)
Pass an item
Item Purchase response
Push a coupon
Aggregate value tokens
Proximity Protocol
Non-electric interaction
TCP/IP

Figure 3-3 Mobile Wallet Scenario

First, a customer purchase value tokens from a value token brokerage.

1) The customer goes to the proximity of a kiosk-like value token charger that is equipped with a wireless proximity communication protocol (e.g., Bluetooth) to purchase value tokens. The value token charger works as a gateway between proximity communication protocol and the Internet infrastructure.
2) The user requests ticket purchase through the user interface. The TMD establishes a secure communication context using the Context Establishment Protocol (Section 4.7) against the value token brokerage, via the value token charger acting as a gateway. At the same time, the TMD and the value token brokerage authenticate each other. Note that since the Context Establishment Protocol is based on the WS-Security [WS-Security], the TMD and the value token brokerage does not have to share a direct TCP/IP connection to keep a secure communication context.
3) The value token brokerage asks the VE to validate attestation it received from the TMD in the process of the context establishment.
4) When the value token brokerage is satisfied with the validation result, the value token brokerage and the TMD can exchange a purchase contract. The contract can be made securely by using the Fair Contract Signing Protocol (Section 6).
5) As the contract progresses, the e-Ticket issuing service may communicate with a bank to verify if the user has enough credit to buy the value tokens.
6) The electrical representation of value tokens is created by the merchant (or the event promoter) in the form of a SAML assertion (Section 4.12), and signed by the merchant to avoid forgery. The value token brokerage installs the value tokens into the TMD by the Remote Data Binding Protocol (Section 5.7), ensuring that the value tokens can be accessed only by the value token brokerage.

After the value tokens are stored in the TMD, the customer purchases an item using the value tokens.

7) The TMD requests a certificate from the VE using the push mode of the Validation Data Protocol. The certificate attests that the current integrity metrics of the TMD is trusted. The request may happen anytime and the certificate is valid as long as the TMD has the same integrity metrics.
8) The customer goes to the proximity of a vending machine, and set up a secure context using the Context Establishment Protocol on top of the proximity protocol. At the same time, the TMD and the vending machine authenticate each other.
9) The TMD and the vending machine exchange purchasing contract using the Fair Contract Signing Protocol (Section 6).
10) The value token is redeemed by the vending machine by the Remote Data redemption Protocol while an item (e.g., a can of soda) is released in return.
11) In addition, the vending machine may push an electric discount coupon to the TMD. The coupon may be a SAML assertion signed by the value token brokerage.
12) The consumed value token is aggregated to the value token brokerage from the vending machine.

The value token brokerage and vending machine are concerned with the following security problems:

Issue/Concern	Mitigation Technique
Alteration of a value token	The value token is digitally signed by the issuer
Forgery of a value token	
Replay attack	Each message in the secure context has sequence number in it, thus the replay attack can be detected.
Duplication of a value token	The Remote Data Binding Protocol ensures that the tickets are encrypted by a key that is known only to the ticket issuer, thus event the device owner cannot access the tickets data.
A value token is used multiple times	The Remote Data Redemption Protocol ensures that the value token is removed from the TMD after it is used.

Issuing a ticket that is not paid for (unfairness)	Purchase takes place in the Fair Contract Signing protocol

The customer is concerned with the following security problems:

Issue/Concern	Mitigation Technique
Hijacking a purchased value tokens over the air	Value token is encrypted by the session secret that is known only to the value token brokerage and the TMD.
Stealing a value token from the TMD	The value token is stored in the protected storage which can be accessed only by the authorized user
Purchasing a value token from an unauthorized ticket issuer (buying a forgery)	Ticket issuer is authenticated before purchase details are transmitted
Being charged for a value token that is never delivered (unfairness)	Purchase takes place in the Fair Contract Signing protocol

3.2.5 Business Workflow Scenario

The Phase-Zero document [**PhaseZero]** identifies several applications in the Business Workflow use-case. In this section, we chose a basic business-to-employee application to exercise a use of the Trusted Mobile Platform protocols in the scenario. In this scenario, we assume that a centralized Workflow Server controls a workflow by generating a form, notifying each employee for approval, and verifying the signature of the approver. Figure 3-4 shows a typical sequence that occurs when an employee requests a reimbursement which will be approved by his manager.

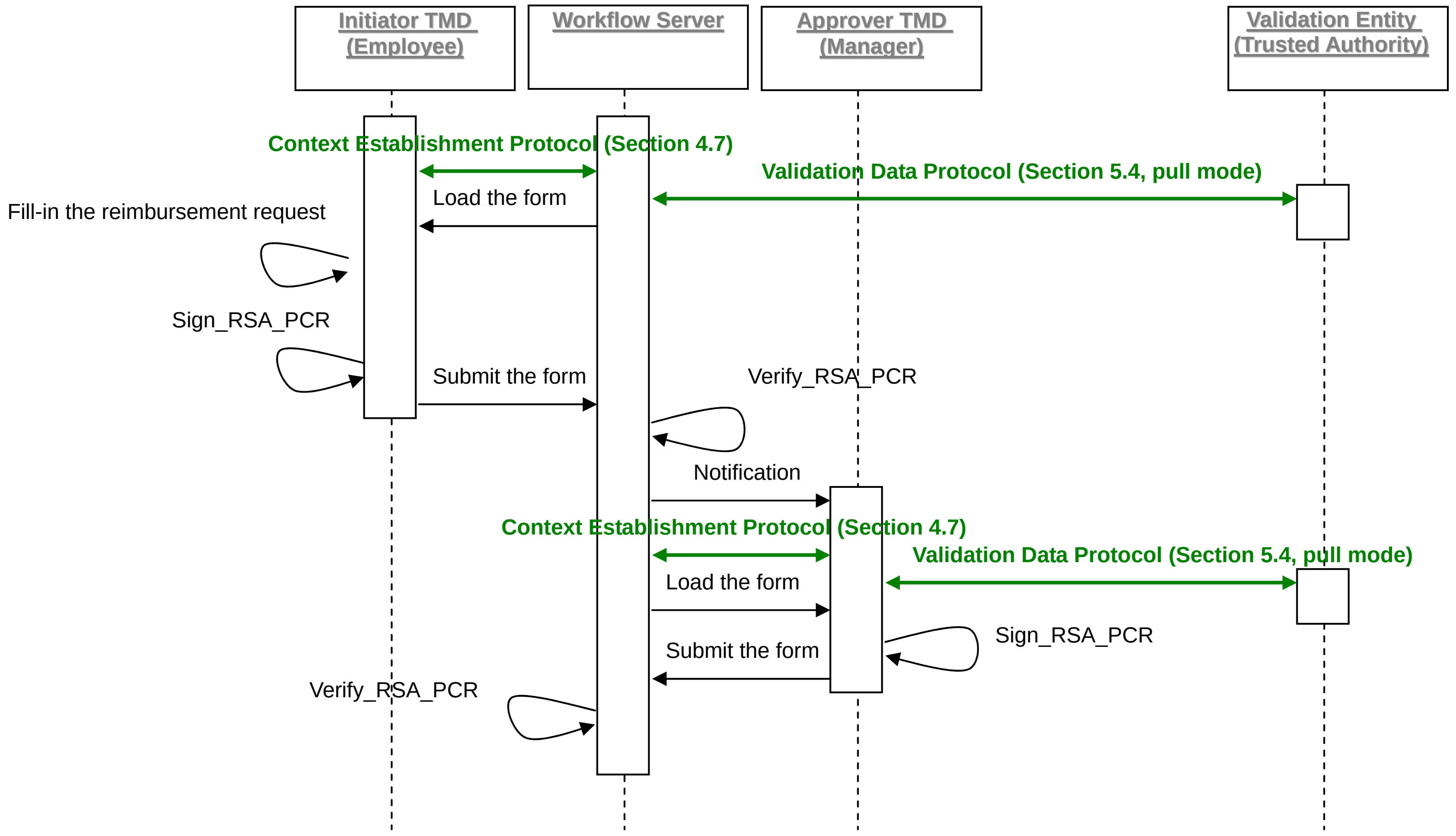

Initiator TMD (Employee)
Workflow Server
Approver TMD (Manager)
Validation Entity (Trusted Authority)
Context Establishment Protocol (Section 4.7)
Validation Data Protocol (Section 5.4, pull mode)
Load the form
Fill-in the reimbursement request
Sign_RSA_PCR
Submit the form
Verify_RSA_PCR
Notification
Context Establishment Protocol (Section 4.7)
Validation Data Protocol (Section 5.4, pull mode)
Load the form
Sign_RSA_PCR
Submit the form
Verify_RSA_PCR

Figure 3-4 Business Workflow Scenario (B2E)

1) First, the employee attempts to initiate a reimbursement request by accessing the Workflow Server. The employee and the Workflow server authenticate each other and setup a secure communication context using the Context Establishment Protocol (Section 4.7).
2) The Workflow Server accesses the VE to validate the identity and the integrity metrics of the employee's Trusted Mobile Device (TMD), which are sent as a part of the Context Establishment Protocol.
3) The client load the request form generated by the Workflow Server.
4) The employee fills-in the form.
5) The employee signs the form using the `Sign_RSA_PCR` operation and submits it. The `Sign_RSA_PCR` operation will sign the timestamp, digest of the form, and the current integrity metrics of the TMD.
6) The Workflow Server verifies the signature by the `Verify_RSA_PCR` operation.
7) The Workflow Server sends a notification to the employee's manager's TMD, who is the first approver in this scenario.
8) The manager's TMD establishes a secure communication context with the Workflow Server.
9) The Workflow Server verifies accesses the VE to validate the identity and the integrity metrics of the manager's TMD, which are sent as a part of the Context Establishment Protocol.
10) The manager's TMD loads the form.
11) The manager signs the form using the `Sign_RSA_PCR` operation and submits it. The `Sign_RSA_PCR` operation will sign the timestamp, digest of the form, the employee's signature, and the current integrity metrics of the manager's TMD.
12) The Workflow Server verifies the manager's signature by the `Verify_RSA_PCR` operation.
13) The process between 7-12 may repeat if there are multiple approver after the first-line manager.

The employee is concerned with the following security problems:

Issue/Concern	**Mitigation Technique**
Violation of privacy by eavesdropping the form	The form is encrypted by the session secret that is exchanged and shared only with the Workflow Server.
Alternation of the form	The form is signed by the employee's attestation signature
Submission to a fake Workflow Server	The Workflow Server and the employee authenticate each other before initiating the transaction.
The manager forgets approving the form, or ignores it pretending there was no submission	The Workflow Server acts as a trusted third party and responsible for keeping track of the workflow. The Workflow Server notifies the manager if the approval was not made for a certain period of time, and notifies his/her superior if no action was taken for long time.
A malicious unauthorized user who has physical access to the TMD sends a request without the employee's consent	It requires the authorization secret to sing the form using the `Sign_RSA_PCR` operation.

The manager is concerned with the following security problems:

Issue/Concern	**Mitigation Technique**
Replay Attack	Since the form and the timestamp are signed, the workflow server can verify the duplication.
Violation of privacy by eavesdropping the form	The form is encrypted by the session secret that is exchanged and shared only with the Workflow Server.
Form submission from a fake employee	The form is signed by the employee's attestation signature.
Alternation of the form	The form, along with the employee's signature and timestamp, is signed by the manager's attestation signature
Notification from a fake Workflow Server	The Workflow Server and the manager authenticate each

	other before initiating the transaction.

The Workflow Server is concerned with the following:

Issue/Concern	Mitigation Technique
Replay Attack	Since the form and the timestamp are signed, the workflow server can verify the duplication.
Integrity of a TMD is compromised	Attestation signature used to establish the secure context and sign the form includes the PCR value when the signature is made, thus the state of the TMD at that point can be verified.
Communicating to a fake employee / manager	The form is signed by the employee's attestation signature.

3.2.6 Unsubscription from a Service and Revocation of an Identity

Figure 3-5 shows a typical sequence of actions that occur when a user unsubscribes from a service.

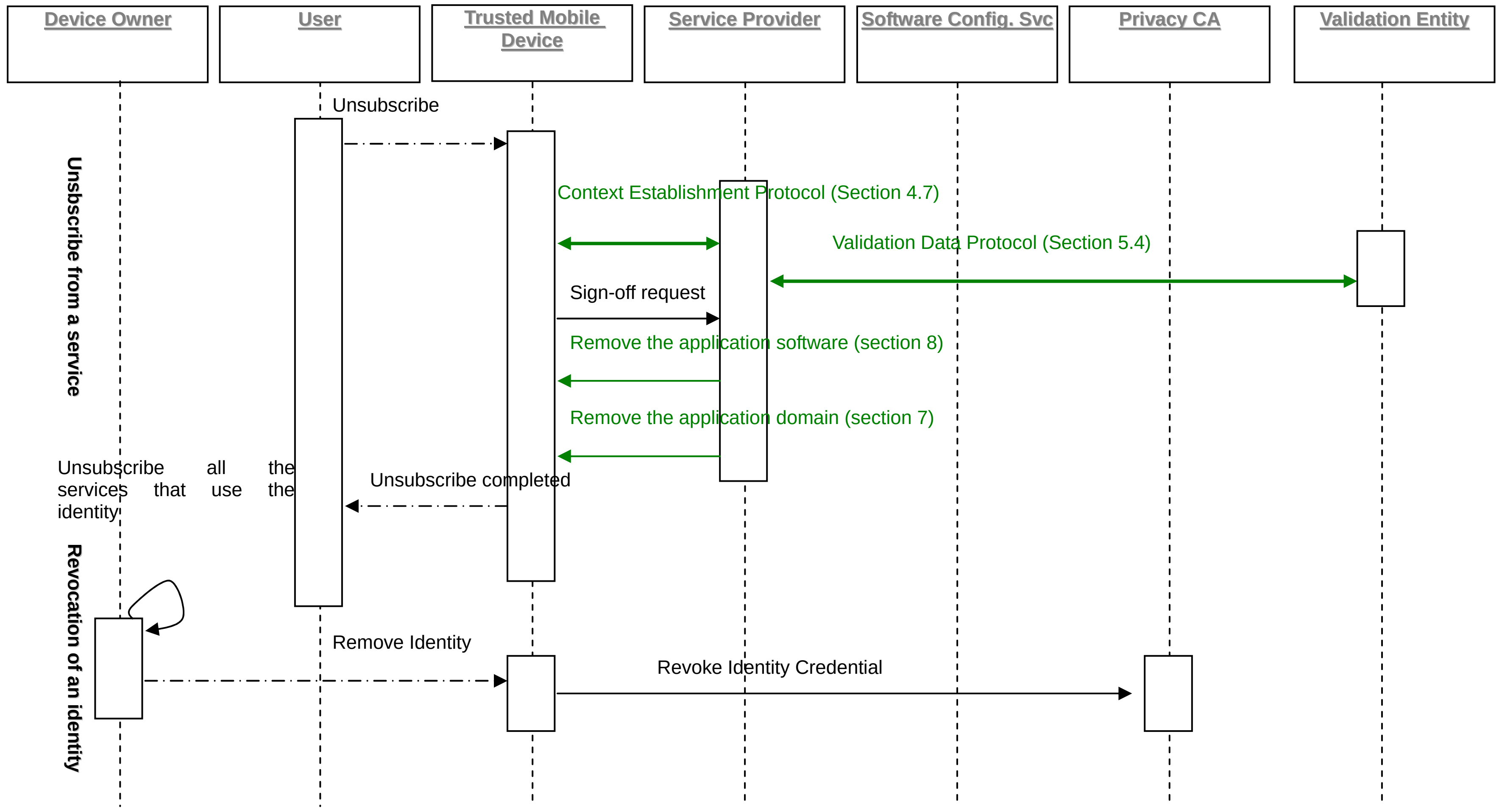

Device Owner
User
Trusted Mobile Device
Service Provider
Software Config. Svc
Privacy CA
Validation Entity
Unsbscribe from a service
Unsubscribe
Context Establishment Protocol (Section 4.7)
Validation Data Protocol (Section 5.4)
Sign-off request
Remove the application software (section 8)
Remove the application domain (section 7)
Unsubscribe all the services that use the identity
Unsubscribe completed
Revocation of an identity
Remove Identity
Revoke Identity Credential

Figure 3-5 Unsubscription from a Service and Revocation of an Identity

1) The user attempts to unsubscribe from a service through the user interface on the device.
2) The TMD sets up a secure session with the service provider using the Context Establishment protocol.
3) The integrity metrics sent to the service provider is validated by the VE and the result is returned to the service provider.
4) The TMD sends an unsubscription request to the service provider.
5) When the service provider is satisfied by the validation result, it remotely removes application software on the TMD. (See section 7).
6) The service provider then removes the application domain associated with its service, and then clears the access control list for the domain. (See section 8)

The device owner may revoke the platform identity to avoid further use. It is anticipated that the user (or the device owner) unsubscribes from all the services that use a platform identity before revoking it.

7) The device owner attempts to revoke the identity through the user interface on the device. The request should be accompanied by the owner authorization secret, so only the legitimate owner can revoke the identity.

The Device User is concerned with the following security problems:

Issue/Concern	Mitigation Technique
A rogue, who has physical access to the device, unsubscribes from a service using the device	The action requires authorization secret on the platform identity by which the user subscribed to the service.
An intermediate pretends to be the service provider and uninstalls the service	The user authenticates the service provider by the Context Establishment Protocol.

The Service Provider is concerned with the following security problems:

Issue/Concern	Mitigation Technique
The core software on the device is compromised	The service provider verifies the integrity metrics in the attestation, which is exchanged in the Context Establishment Protocol.
An intermediate pretends to be the device to unsubscribe from a service	The service provider authenticates the requester by the Context Establishment Protocol, in which attestation is exchanged.

4 WS-Security Profiles

4.1 Web Services

Trusted Mobile Devices are used as client platforms for distributed applications. Well-known applications such as Web browsers and e-mail clients still utilize existing protocols and data formats such as HTTP, HTML, POP3, SMTP, and MIME. General distributed applications require a standard framework for distributed computing. Distributed computing frameworks such as DCE, CORBA, and DCOM are commonly used within today's network infrastructure. Recently, Web services have introduced yet another distributed computing framework that enables loosely-coupled applications, supporting a wide variety of platforms (including Microsoft's Windows platforms and Unix-based platforms). Due to the ubiquitous nature of Web services capabilities, it is well anticipated that Web services will be widely used as a preferred distributed computing framework within the next few years. The fundamental nature that the trusted mobile devices are programmable distributed computing platforms, they should be capable of communicating with these servers (services).

From a security requirement point of view, *authentication, confidentiality, and integrity* are currently provided by Transport Layer Security (TLS), under the assumption that there is a direct TCP/IP connection between the client platform and the server. On the other hand, a Class 3 Trusted Mobile Platform provides additional protection in the following used cases:

When there is no direct TCP/IP connection between the client and the server. For example, the network operator has a gateway between the public Internet and their private access network. With the weaker operator domain trust assumption (see Section 2.1), Trusted Mobile Devices will have to depend on the message layer security rather than the transport layer security.

When security requirements are beyond authentication, confidentiality, and integrity. Non-repudiation of a message is one such example. Another example is when a message needs to be associated with its proper authorization information. Trusted Mobile Platforms also need to exchange the platform integrity claims. These security requirements are essential for realizing the e-ticket scenario and the B2B scenario described in the Phase Zero document **[PhaseZero]**.

When the communications pattern are not synchronous and peer-to-peer. Asynchronous communications require message-level protection. Also multi-party communications would require additional security mechanism(s).

The Web Services Security specification **[WS-Security]** provides flexible, extensible, and interoperable framework to the security protocol. The Trusted Mobile Platform takes advantages of WS-Security to support the Trusted Mobile Platform's High Security Option. However, WS-Security as a whole is a large and complex specification, not all of which i practical to a resource-constrained mobile device. This section defines profiles to WS-Security; so that it can be efficiently implemented on such devices while preserving interoperability.

4.2 Namespaces

The following namespaces are used in this document. Unless explicitly stated otherwise, the prefixes in this table are assumed to be bound to the corresponding namespace URI's **[NameSpace] [URI]**.

Prefix	**Namespace**	**Defined in**
S	`http://www.w3.org/2001/12/soap-envelope`	**[SOAP11]**

ds	`http://www.w3.org/2000/09/xmldsig#`	**[XML-Signature]**
xenc	`http://www.w3.org/2001/04/xmlenc#`	**[XML-Encryption]**
wsse	`http://schemas.xmlsoap.org/ws/2002/06/secext`	**[WS-Security]**
wst	`http://schemas.xmlsoap.org/ws/2002/12/secext`	**[WS-Trust/WS-SecureConversation]**
wsa	`http://schemas.xmlsoap.org/ws/2003/03/addressing`	**[WS-Addressing]**
wsu	`http://schemas.xmlsoap.org/ws/2003/06/utility`	**[WS-Security]**
saml	`urn:oasis:names:tc:SAML:1.0:assertion`	**[SAML]**
tmpa	`http://schemas.trusted-mobile.org/protocol/2003/06/ws`	**[TMP-Protocol]**

This specification explicitly REQUIRES supporting SOAP 1.1 **[SOAP11]** because it is the basis for the suite of Web Services Security specifications as of June 2003. However, we anticipate that SOAP 1.2 **[SOAP12]** will also be used. Therefore, it is RECOMMENDED that implementations support SOAP 1.2 and allow SOAP 1.1 messages according to the version transition rules described in Appendix A of **[SOAP12-Part1]**.

4.3 WS-Security [WS-Security]

The types of security tokens that MUST be supported are listed in the following table.

QName	Description
`wsse:UsernameToken`	Username, optionally with a password **[WS-Security]**.
`wsse:BinarySecurityToken`	Binary security token **[WS-Security]**. See the next table for the details of subtypes.
`saml:Assertion`	SAML Assertion **[SAML]**. See **[WS-SAML]** for the details of this binding

`wsse:UsernameToken`	Used for asserting the user's identity. This token is simple to implement, and useful in certain situations where impersonation is required (for example, a gateway impersonates the end user).
`wsse:BinarySecurityToken`	A generic type that may have different types of existing security tokens, including X.509 certificates and TCG-specific credentials.
`saml:Assertion`	Used for carrying authorization information.

The types of binary security token that MUST be supported in this specification are listed in the following table.

QName	Description
`wsse:X509v3`	X.509 v3 certificate **[WS-X509]**.
`tmpa:TpmIdentityCredential`	TCG identity credential. See Section 5.3 for details.
`tmpa:TCGValidationData`	TCG validation data token. See Section 5.4 for details.

4.4 SOAP Message Structure

The following shows the structural overview of a SOAP envelope conforming to this profile.

```
<S:Envelope xmlns:S=”…”>
 <S:Header>
  <wsse:Security>
   <wsse:BinarySecurityToken> … </wsse:BinarySecurityToken>
   <xenc:EncryptedKey> … </xenc:EncryptedKey>
   <ds:Signature> … </ds:Signature>
  </wsse:Security>
  <wsa:From wsu:Id=”…”> … </wsa:From>
  <wsa:To wsu:Id=”…”> … </wsa:To>
  <wsa:MessageId wsu:Id=”…”> … </wsa:MessageId>
  <wsu:Timestamp wsu:Id=”…”> … </wsu:Timestamp>
 </S:Header>
 <S:Body wsu:Id=”…”>
  <xenc:EncryptedData> … </xenc:EncryptedData>
 </S:Body>
</S:Envelope>
```

4.4.1 Message Integrity

For message integrity, a SOAP message SHOULD be digitally signed. When a SOAP message is signed, the SOAP Body element MUST always be signed. Besides, the following message header blocks MUST be present and signed. Signing these header blocks ensures that a captured message cannot be used for other purposes (e.g., sending it to other destinations, spoofing the message sender, or replaying the message). These header blocks MUST appear after the *`ds:Signature`* element that signs them so that the receiving endpoint can process them efficiently. In addition, the references to these header blocks MUST be done through intra-document references (i.e., URI=“#...”).

Header Block	**Description**
`wsa:From`	The source endpoint of this message **[WS-Addressing]**
`wsa:To`	The destination endpoint of this message **[WS-Addressing]**
`wsu:Timestamp`	Timestamp of the message creation **[WS-Security]**
`wsa:MessageID`	A unique identifier of the message **[WS-Addressing]**

In order to ensure message freshness, the endpoint that received a signed message SHOULD keep some form of a message *`Timestamp`* and *`MessageID`* in the resident cache for a duration that is defined by the endpoint’s policy. When receiving a new message, the recipient SHOULD look up the cache for detecting replay attacks. If a message with the same *`MessageID`* is found in the cache, the recipient MUST cease processing the newly received message and MAY report the error to the sender. The recipient SHOULD reject the message if the time-stamp has expired for the given cache duration time.

The signature key may be either the sender's private key or a session context secret (an HMAC key that is derived from a session context) if such a context exists (See 4.7 for establishing a context).

4.4.2 Message Confidentiality

When a SOAP message is encrypted, the content of the SOAP Body element MUST always be encrypted.

If there is a shared session secret (See 4.7 for establishing a context), the message MAY be encrypted by the secret or a key derived from the secret. If there is no such shared secret, an encryption key needs to be generated and wrapped in a *<xenc:EncryptedKey>* element, which is protected by the recipient's public key.

4.4.3 Message Authentication

For proper message authentication, a message must be signed and encrypted as described in this section. There are two scenarios how mutually-authenticated message exchange can be performed with this specification.

4.4.3.1 Stateless message exchange

If there is a no session secret shared by the parties, each message needs to be signed by the sender's private key and encrypted by the receiver's public key (through encryption by a temporary symmetric key that is encrypted by the receiver's public key). With the signature, which includes the sender's name, recipient's name, message Id, timestamp, and the message body in its scope, the receiver can verify that the message is actually coming from the sender and being addressed to the receiver, that the message has not been modified during the transmission, and that the message is not a replay of a previous message.

With the encryption, the sender can be confident that the message is only read by the intended receiver. Note that message authentication does not guarantee the delivery of the message. If a message delivery confirmation is necessary, the application needs to use a request-response pattern. In this case, the *<wsa:RelatesTo>* header block **[WS-Addressing]** SHOULD be used in the reply message (which is also signed and encrypted) so that the sender can verify the delivery of the first message.

4.4.3.2 Stateful message exchange

If there is a session secret shared by the parties, each message can be signed (i.e., HMAC'ed) and encrypted by the session secret or keys that are derived from the session secret. The session secret is shared by performing one of the session establishment protocols described in Section 4.7. Since the session secret is shared only if the both parties are properly authenticated and nobody else knows the session secret (unless one of the parties release the secret to others), the use of the session secret is strongly tied to the authenticity of the parties. Therefore, the messages signed and encrypted by the session secret or keys that are derived from the session secret can be considered to be authenticated. That is, the sender can be confident that the only entity who can read the message is the receiver and the receiver can be confident that the message is coming from the sender. In addition, message integrity and message confidentiality are maintained.

4.4.4 Derived Keys

When a session secret is used for protecting messages, it is RECOMMENDED that actual keys to be used for signing and encryption are derived from the session key using the mechanism defined in Section 5 of WS-SecureConversation **[WS-SecureConversation]**. In such a case, one <`wsse:DerivedKeyToken`> for each derived key is included in the <`wsse:Security`> header block. This <`wsse:DerivedKeyToken`> element in turn refers to the security context token being used.

4.5 Minimalist Profile of WS-Security

Class 3 Trusted Mobile Devices are RECOMMENDED to support the Minimalist Profile of WS-Security **[Minimal-WS-Security]**. Due to the limitations of mobile devices, limited resources in terms of processing power, memory, and power consumption, it is important to keep in mind that the protocol is implemented as a one-pass process. The Minimalist Profile allows the receiving endpoint to perform WS-Security processing in a streaming manner.

A Trusted Mobile Device that supports the Minimalist Profile of WS-Security SHOULD advertise its capability so that the communicating party can create messages conforming to the Minimalist Profile of WS-Security. How this advertisement should be conducted is out of scope of this specification. Any endpoint that potentially communicates with a Trusted Mobile Platform device is RECOMMENDED to always to generate SOAP messages conforming to the Minimalist Profile.

4.6 WS-Trust

WS-Trust **[WS-Trust]** defines a protocol between endpoints and a security token service. It is RECOMMENDED for Class 3 Trusted Mobile Platforms to support WS-Trust in the following situations:

For requesting and receiving TCG-related credentials to the device. See Section 5 for the detailed use of WS-Trust for this purpose.

For establishing a security context between two endpoints. (See Section 4.7 for the details).

For obtaining and validating certificates. See Section 7.3 for more on key management issues. If the infrastructure (i.e., CA, directory, etc.) supports WS-Trust, the implementation is RECOMMENDED to use WS-Trust.

For exchanging other application specific tokens such as SAML tokens for authorization.

WS-Trust is can be used for any one of the above features. If the device is to use the above features, WS-Trust should be implemented. Since WS-Trust is not a standard document as of June 30, 2003, we expect that WS-Trust will evolve in the future. Implementations should be prepared to adopt to these changes.

4.7 Context Establishment

When two parties exchange multiple messages in succession, it is desirable that they exchange a common session key and use this key for signing and encrypting the messages instead of using expensive public key operations. This subsection describes the protocols for establishing and destroying security context. It shows how these two endpoints authenticate each other and share a fresh session secret. We propose the following three protocols depending on requirements of the situation.

1. **Diffie-Hellman key exchange** *(Section 4.7.1)*
   When the both parties want to use a signature-only key for authentication, this protocol can be used. An example of such situation is that the parties use attestation keys for both authentication and attestation.

2. **Public key based key exchange** *(Section 4.7.2)*
   When one party has a TPM with a capability of generating a symmetric key inside the TPM, this protocol can be used for securing the session key.

3. **Symmetric key based key exchange** *(Section 4.7.3)*
   When the parties already share a symmetric key (session secret) and want to use it for establishing a new context, this protocol can be used. Also this protocol can be used for verifying the proof of possession of the shared session secret.

For a Class 3 Trusted Mobile Device, it is RECOMMENDED that one of the above protocols is implemented.

A session is represented as a security context token **[WS-SecureConversation]**. During the session, the communicating parties keep the security context token along with the session secret that is exchanged using one of the protocols described below. The security context token has a global Id (URI) that is unique to both parties. When referring to the session secret, this URI SHOULD be used in a <`wsse:SecurityTokenReference`> element.

4.7.1 Diffie-Hellman Key Exchange

4.7.1.1 Protocol Overview

The overview of this protocol is shown in Figure 4-1 DH-based Context Establishment Protocol. This protocol is based on ephemeral Diffie-Hellman key exchange with digital signatures for authentication.

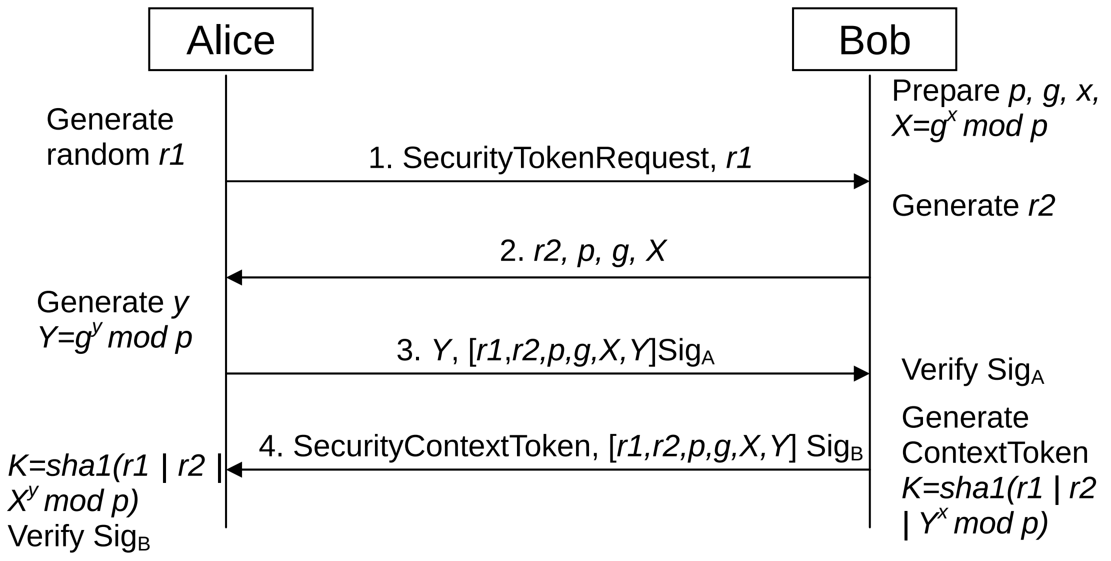

Alice
Bob
Generate random $r1$
Prepare $p, g, x, X=g^x \bmod p$
1. SecurityTokenRequest, $r1$
Generate $r2$
2. $r2, p, g, X$
Generate y
$Y=g^y \bmod p$
3. Y, $[r1,r2,p,g,X,Y]\mathrm{Sig}_A$
Verify Sig_A
Generate ContextToken
4. SecurityContextToken, $[r1,r2,p,g,X,Y]\ \mathrm{Sig}_B$
$K=sha1(r1 \mid r2 \mid X^y \bmod p)$
Verify Sig_B
$K=sha1(r1 \mid r2 \mid Y^x \bmod p)$

Figure 4-1 DH-based Context Establishment Protocol

4.7.1.2 Message 1 Syntax and Processing

The requester, Alice, first generates a random number *r1* and sends it to the responder, Bob. The following shows the syntax of Message 1 sent from Alice to Bob.

```
<wst:RequestSecurityToken>
   <wst:TokenType>tmpa:SessionContextToken</wst:TokenType>
   <wst:RequestType>wst:ReqIssue</wst:RequestType>
   <tmpa:Requester>
      <wsse:Nonce>FZEkQjaigph/bqYNlGJYpMQ6/ds=</wsse:Nonce>
   </tmpa:Requester>
</wst:RequestSecurityToken>
```

The following describes elements and attributes used in the <`wst:RequestSecurityToken`> element.

`/RequestSecurityToken/TokenType`

This mandatory element represents the type of the security token being requested and MUST contain the qname “tmpa:SessionContextToken”.

`/RequestSecurityToken/RequestType`

This mandatory element represents the type of this request and MUST contain the QName “wst:ReqIssue”.

`/RequestSecurityToken/tmpa:Requester`

This mandatory element contains information on the requester.

`/RequestSecurityToken/tmpa:Requester/Nonce`

This mandatory element contains the base64-encoded random number *r1*.

/RequestSecurityToken/tmpa:Requester /@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/RequestSecurityToken/tmpa:Requester /{any}

This is an extensibility mechanism to allow additional elements to be used.

If Bob receives a message that does not conform to the syntax of Message 1, he MAY report the syntax error to Alice using the SOAP fault code defined in 4.8.

4.7.1.3 Message 2 Syntax and Processing

On receipt of this message, the responder (Bob) generates its own random number *r2*. In addition, Bob needs to have prepared a Diffie-Hellman key pair, *X* and *x* such that $X=g^x \bmod p$. This Diffie-Hellman key generation can be done at any time. (e.g., Bob's Diffie-Hellman key may be generated at the startup time. Bob responds with this Diffie-Hellman key along with the freshly generated random number *r2*).

The following is the syntax of Message 2 from Bob to Alice.

```
<wst:RequestSecurityTokenResponse>
  <wst:SignChallenge>
    <wst:Challenge>
      <tmpa:Responder>
        <wsse:Nonce>FZEkQjaigph/bqYNlGJYpMQ6/ds=</wsse:Nonce>
        <tmpa:DiffieHellmanPublic> … </tmpa:DiffieHellmanPublic>
      </tmpa:Responder>
    </wst:Challenge>
  </wst:SignChallenge>
</wst:RequestSecurityTokenResponse>>
```

The following describes elements and attributes used in Message 2.

/RequestSecurityTokenResponse/SignChallenge

This mandatory element represents that Bob is requesting Alice's signature.

/RequestSecurityTokenResponse/SignChallenge/Challenge

This mandatory element specifies the element that needs to be signed by Alice.

/RequestSecurityTokenResponse/SignChallenge/Challenge/tmpa:Responder

This mandatory element represents Bob's cryptographic parameters, including the Diffie-Hellman key and the random number *r2*.

/RequestSecurityTokenResponse/…/tmpa:Responder/Nonce
This mandatory element contains the base64-encoded random number *r2*.

/RequestSecurityTokenResponse/…/tmpa:Responder/tmpa:DiffieHelmanPublic
This mandatory element represents Bob's Diffie-Hellman public key. See 4.7.4 for the details of this element

/RequestSecurityTokenResponse/…/tmpa:Responder /@{any}
This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/RequestSecurityToken Response/…/tmpa:Responder /{any}
This is an extensibility mechanism to allow additional elements to be used.

If Alice receives a message that does not conform to the syntax of Message 2 as a response to her Message 1, she MUST abort the protocol and MUST report the syntax error to Bob using the SOAP fault code defined in 4.8.

4.7.1.4 Message 3 Syntax and Processing

After receiving Message 2 from Bob, Alice generates her ephemeral Diffie-Hellman key pair, *y* and *Y* such that $Y=g^y \bmod p$. Then she signs *r1, r2*, Bob's public key, and Alice's public key using her authentication key (for example, her AIK). The message sent back to Bob, Message 3, has the following syntax. This message MUST be signed by Alice's authentication key using the syntax described in 4.4 SOAP Message Structure.

```
<wst:RequestSecurityTokenResponse>
  <wst:SignChallengeResponse>
    <tmpa:Requester>
      <wsse:Nonce>FZEkQjaigph/bqYNlGJYpMQ6/ds=</wsse:Nonce>
      <tmpa:DiffieHellmanPublic> … </tmpa:DiffieHellmanPublic>
    </tmpa:Requester>
    <tmpa:Responder>
      <wsse:Nonce>FZEkQjaigph/bqYNlGJYpMQ6/ds=</wsse:Nonce>
      <tmpa:DiffieHellmanPublic> … </tmpa:DiffieHellmanPublic>
    </tmpa:Responder>
  </wst:SignChallengeResponse>
</wst:RequestSecurityTokenResponse>
```

The following describes elements and attributes used in Message 3.

/RequestSecurityTokenResponse/SignChallengeResponse
This mandatory element represents Alice's response to Bob's signing challenge.

/RequestSecurityTokenResponse/SignChallengeResponse/tmpa:Requester

This mandatory element contains cryptographic parameters of Alice.

`/RequestSecurityTokenResponse/…/tmpa:Requester/Nonce`
This mandatory element contains the base64-encoded random number *r1*.

`/RequestSecurityTokenResponse/…/tmpa:Requester/tmpa:DiffieHelmanPublic`
This mandatory element represents Alice's Diffie-Hellman public key. See 4.7.4 for the details of this element

`/RequestSecurityTokenResponse/SignChallengeResponse/tmpa:Responder`
This mandatory element contains cryptographic parameters of Bob.

`/RequestSecurityTokenResponse/…/tmpa:Responder/Nonce`
This mandatory element contains the base64-encoded random number *r2*.

`/RequestSecurityTokenResponse/…/tmpa:Responder/tmpa:DiffieHelmanPublic`
This mandatory element represents Bob's Diffie-Hellman public key. See 4.7.4 for the details of this element

Upon receipt of Message 3, Bob MUST verify Alice's signature on the message. If the syntax is incorrect, the signature does not verify, or at that time Bob can decide whether that signature key can not be trusted according to his policy, Bob MUST abort the protocol session. Bob MAY also report the error using the SOAP fault code defined in 4.8.

4.7.1.5 Message 4 Syntax and Processing

If Bob is satisfied with Message 3, he constructs a security context token, generates the session key K such that $K=sha1(r1 \mid r2 \mid Y^x \bmod p)$ where | denotes string concatenation.

Bob then sends the final message, Message 4, back to Alice. The syntax of this message is as follows. This message MUST be signed by Bob's authentication key using the syntax described in 4.4 SOAP Message Structure

```
<wst:RequestSecurityTokenResponse>
  <wst:RequestedSecurityToken>
      <wst:SecurityContextToken> … </wst:SecurityContextToken>
  <wst:RequestedSecurityToken>
  <tmpa:Requester> … </tmpa:Requester>
  <tmpa:Responder> … </tmpa:Responder>
</wst:RequestedSecurityToken>
```

The following describes elements and attributes used in Message 4.

`/RequestSecurityTokenResponse/RequestedSecurityToken`
This mandatory element represents the security context token returned by Bob. This element MUST contain a single element that is the security context token generated by Bob.

/RequestSecurityTokenResponse/ tmpa:Requester

This mandatory element contains cryptographic parameters of Alice, including the Diffie-Hellman key and the random number *r1*.

/RequestSecurityTokenResponse/tmpa:Responder

This mandatory element contains cryptographic parameters of Bob, including the Diffie-Hellman key and the random number *r2*.

Upon receipt of Message 4, Alice MUST verify Bob's signature on the message. If the syntax is wrong, the signature does not verify, or Alice can decide that the signature key cannot be trusted according to her policy, Alice MUST abort the protocol session. Alice also MUST report the error using the SOAP fault code defined in 4.8

4.7.2 Public Key-Based Key Exchange

The general description of this protocol is given in Section 7.3.2. Binding of this protocol to WS-Security is TBD.

4.7.3 Symmetric Key-Based Key Exchange

The general description of this protocol is given in Section 8.3.4. Binding of this protocol to WS-Security is TBD.

4.7.4 Tearing Down Session Context

For tearing down an existing session context, we need a way to revoke the security context token representing the session context. Unfortunately the current WS-Trust draft does not define how to revoke an already-issued security token. We temporary define the following request type of WS-Trust for our purpose. We expect that WS-Trust will define a mechanism to do this, and this specification will be subsumed by the new WS-Trust definition once it is defined.

QName	Description
`tmpa:ReqRevoke`	Revoke request of security token

The following illustrates the syntax of a session tear-down request.

```
<wst:RequestSecurityToken>
   <wst:TokenType>tmpa:SessionContextToken</wst:TokenType>
   <wst:RequestType>tmpa:ReqRevoke</wst:RequestType>
   <wst:Base>
      <wst:Reference URI=””/>
   </wst:Base>
</wst:RequestSecurityToken>
```

The following describes elements and attributes used in this request.

/RequestSecurityTokenResponse/TokenType

The value of this mandatory element MUST be the qname `“tmpa:SessionContextToken”..`

/RequestSecurityTokenResponse/RequestType
The value of this mandatory element MUST be the qname "`tmpa:reqRevoke`".

/RequestSecurityTokenResponse/Base
This mandatory element contains a reference to the session context being revoked.

This message MUST be signed (HMAC'ed) using a key derived from the session context that is being revoked.

On the successful revocation of the security token, the security token service MUST return a message with an empty <*wst:RequestSecurityTokenResponse*> as the body. If the revocation is not successful, a SOAP fault message using the fault codes in Section 4.8 SHOULD be returned. In both cases, the reply message MUST be signed using a key derived from the current session context. Upon successful completion of this exchange, both parties MUST invalidate the security context token (e.g., removing it from the memory).

4.7.5 Diffie-Hellman Parameter Element

The Diffie-Hellman public key parameters are represented by the following syntax. Note that when Alice sends her public key using this syntax, the modulus and the generator are OPTIONAL mainly due to the fact that these values are the same, as with Bob's values. All the parameters MUST be aligned to byte boundaries (i.e., their bit length is multiple of 8).

```
<tmpa:DiffieHellmanPublic>
  <tmpa:Modulus>PoBmL7mRw/GI…3JSyzhDpRlY=</tmpa:Modulus>
  <tmpa:Generator>dNCglMprbcYT…OCrLNczvAm=</tmpa:Generator>
  <tmpa:Public>w/GFShuO2SIXp…65QAw=</tmpa:Public>
</tmpa:DiffieHellmanPublic>
```

The following describes elements and attributes used in `<tmpa:DiffieHellmanPublic>` element.

/DiffieHellmanPublic/Modulus
This element contains the base64-encoded modulus (*p*) of the key. This element is mandatory when Bob sends its key to Alice. This element is optional when Alice sends her key to Bob.

/DiffieHellmanPublic/Generator
This element contains the base64-encoded generator (*g*) of the key. This element is mandatory when Bob sends its key to Alice. This element is optional when Alice sends her key to Bob.

/DiffieHellmanPublic/Public
This mandatory element contains the base64-encoded public key part (*X or Y*) of the key.

4.8 SOAP Fault

Endpoints MUST use the SOAP fault codes defined in WS-Security **[WS-Security]** and WS-Trust **[WS-Trust]** when reporting errors. The errors relevant to this specification are shown in the following table.

Error that occurred	***Faultcode***
The request was invalid or malformed	`wst:InvalidRequest`
Authentication Failed	`wsse:FailedAuthentication`
The specified request failed	`wst:RequestFailed`
Security token has been revoked	`wsse:InvalidSecurityToken`
Insufficient Digest Elements	`wst:AuthenticationBadElements`
The specified `RequestSecurityToken` is not understood.	`wst:BadRequest`

4.9 XML Encryption

WS-Security requires the use of W3C XML Encryption **[XML-Encryption]**. Therefore, a Class 3 Trusted Mobile Device MUST implement XML Encryption. In addition, it is RECOMMENDED that a Class 3 Trusted Mobile Device also provide an API so that applications can use XML Encryption for application-level confidentiality. The algorithms that MUST be supported are summarized in the following table. These algorithm URIs are defined in **[XML-Encryption]**.

Algorithm URI	**Description**
`http://www.w3.org/2001/04/xmlenc#aes128-cbc`	AES 128 bit
`http://www.w3.org/2001/04/xmlenc#rsa-1_5`	RSA v1.5 key transport
`http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`	RSA OAEP key transport

In addition, it is RECOMMENDED that the device also supports Triple DES (http:www.w3.org/2001/04/xmlenc#tripledes-cbc) for interoperability with existing applications.

4.10 XML Digital Signature

WS-Security requires the use of W3C XML Signature **[XML-Signature]**. Therefore, a Class 3 Trusted Mobile Device MUST implement XML Signature. In addition, it is RECOMMENDED that a Class 3 Trusted Mobile Devices also provide an API so that application can use XML Signature for application-level integrity and non-repudiation. The algorithms that MUST be supported are summarized in the following table. These algorithm URIs are defined in **[XML-Signature]**. Support for other algorithms defined in **[XML-Signature]** is RECOMMENDED but not required by this specification.

Algorithm URI	**Description**
`http://www.w3.org/2000/09/xmldsig#sha1`	SHA1 digest

`http://www.w3.org/2000/09/xmldsig#base64`	Base64 Encoding
`http://www.w3.org/2000/09/xmldsig#hmac-sha1`	HMAC-SHA1 **[RFC2014]**
`http://www.w3.org/2000/09/xmldsig#rsa-sha1`	RSA signature (PKCS#1 v1.5) with SHA1
`http://www.w3.org/2000/09/xmldsig#enveloped-signature`	Enveloped signature transform
`http://www.w3.org/2001/10/xml-exc-c14n#`	Exclusive canonicalization **[EXC-C14N]**
`http://www.trustedcomputinggroup.org/algorithms/xmlsig-pcr`	TCG signature with PCR values (*see Section 5.5*)

Note #1: Although DSA algorithm is mandated by XML Signature, it is optional in Trusted Mobile Platform because TPM may not have DSA signing capability.

Note #2: Support for PKCS#1 v1.5 is required because it is mandatory for TPM defined in **[TCG]**. If the TPM supports RSA-PSS, the platform MUST support RSA-PSS. It is RECOMMENDED to support RSA-PSS because they are considered to be more secure than PKCS#1 v1.5.

4.11 Use of TPM for Cryptographic Operations

The Trusted Mobile Device has a hardware module (TPM) that provides secure cryptographic operations. In this specification, it is RECOMMENDED that the TPM's capability is used whenever appropriate in the following situations.

Random number generation
TPM provides h/w-based random number generation, which is considered to be more secure than any known software implementation. It is RECOMMENDED that whenever a random number is needed, the TPM's PRNG is used (e.g., generation of temporary symmetric keys, generation of initialization vectors for encryption, generation of random numbers in the session establishment protocols).

Digital signature
When a digital signature is required, it can be done within the TPM. This allows the signing key to be very well protected inside the TPM, thus enhancing the security.

Attestation signature
As a special case of digital signature, one may use `TPM_QUOTE` so that (1) the signature is done via the AIK which is protected by the TPM, thus, enhancing security, and (2) the signature contains the hash of selected PCR values so that the signature is bound to a particular integrity metric of the platform. If an attestation signature is used in the context establishment protocol (4.7), the session is bound to the platform's state.

Protecting session secret
The session secret (*K*) can also be protected using the TPM. After the session secret is exchanged, it is stored in the TPM. Every so often, the session key is retrieved and used for generating actual encryption keys and HMAC keys. For example, some policy may allow that a session lasts 24 hours, but for each message a temporary encryption key and an HMAC key is generated from the session secret and the message id (within the session). This way, it is harder to hijack a session by stealing the session secret.

4.12 Security Assertion Markup-Language (SAML)

The SAML Assertions **[SAML]** is a prerequisite for the fair contract signing protocol (Section 6). If the device is to use the fair contract singing protocol, SAML assertions MUST be supported. On the other hand, the SAML protocol is subsumed by WS-Trust and thus, the use of the SAML protocol SHOULD be avoided.

4.13 Access Control Policy Language (XACML)

In this specification, it is RECOMMENDED to use XACML **[XACML]** for the access control policy language. An example for using XACML is provided in section 8.5.4.

4.14 Schema

The schema for the XML syntax used in this section is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:wsse=http://schemas.xmlsoap.org/ws/2002/06/secext
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
  xmlns:tmpa="http://schemas.trusted-mobile.org/protocol/2006/06/ws"
targetNamespace="http://schemas.trusted-mobile.org/protocol/2006/06/ws
">
    <xsd:element name="Requester">
      <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="wsse:Nonce"/>
            <xsd:element ref="tmpa:DiffieHellmanPublic"
                                   minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="Responder">
      <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="wsse:Nonce"/>
            <xsd:element ref="tmpa:DiffieHellmanPublic"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="DiffieHellmanPublic">
      <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="tmpa:Modulus" type="xsd:base64Binary"
                                  minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="tmpa:Generator" type="xsd:base64Binary"
                                  minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="tmpa:Public" type="xsd:base64Binary"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

5 Attestation and Supporting Infrastructure

5.1 Introduction

TCG (Trusted Computing Group) is an industry specification that enables trust in computing platforms. The main specification **[TCG]** defines a trusted subsystem that provides useful security capabilities, structures and APIs for utilizing these capabilities from enhanced operating systems and applications to compose a secure computing environment, and trust entities that supports that environment.

A number of credentials and certificates defined in **[TCG]** play an important role in making a trusted platform interact with other entities. These certificates are the evidences that the design and implementation of the system are appropriate and conforming to the specification, a certain identity is derived from a genuine security subsystem called a Trusted Platform Module (TPM), and a certain platform configuration is known and thus considered to be trusted. However, the way of exchanging the certificates is not defined in **[TCG]**. Since some credentials are created and transmitted at the time of operation, protocols for certificates exchange are necessary.

Another kind of protocol is needed when TCG functions are accessed by remote entities. For example, one very important feature of a trusted platform is to support the integrity measurement and reporting capability, which enables the platform to provide mechanisms for cryptographically reporting the current hardware and software configuration of a computer device to challengers.

This specification defines a set of protocols that work among Trusted Mobile Devices, services, and trust entities. Since interoperability is essential for attestation-based trust infrastructure to be widely used in various network environments, the protocols are designed based on the Web Services Security specifications.

The following namespace is used in this document.

Prefix	**Namespace**	**Defined in**
`tmp`	http://schemas.trusted-mobile.org/protocol	This document
`wsse`	http://schemas.xmlsoap.org/ws/2002/06/secext	**[WS-Security]**
`wst`	http://schemas.xmlsoap.org/ws/2002/12/secext	**[WS-Trust]**
`wsu`	http://schemas.xmlsoap.org/ws/2002/07/utility	**[WS-Security]**

5.2 Protocols Overview

Figure 5-1 illustrates the scope of this section.

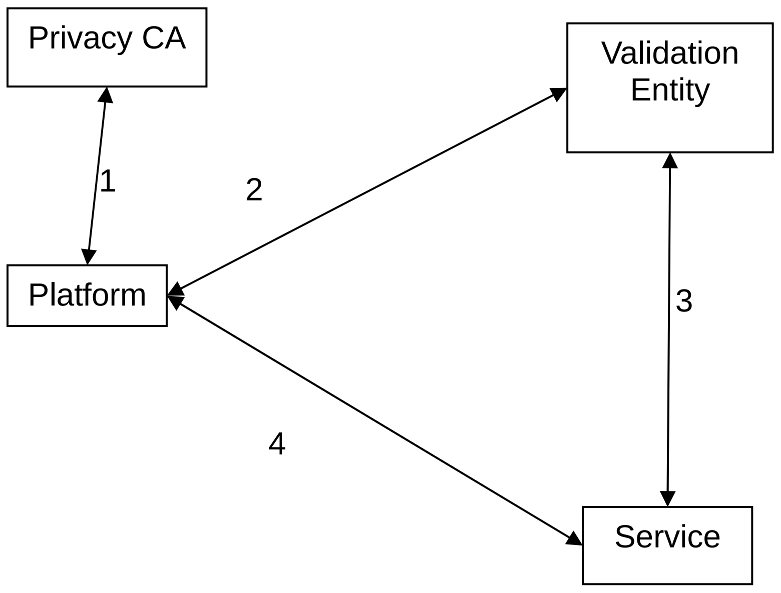


Figure 5-1 Protocol Overview

1. **Privacy CA protocol**	between the platform and the privacy CA. This protocol is used to request a TPM identity certificate for the identity created inside the TPM on the platform. To preserve the pseudonymity of the identity, special request/response message formats are defined in **[TCG]**.
2. **Validation data protocol**	between the platform and the validation entity. This protocol is used by the platform to request a validation certificate for the current platform configuration, which is then pushed to the service along with the messages in the protocol 4 and 5.
3. **Validation data protocol**	between the service and the validation entity. This protocol is used by the service to ask for the validation of the configuration of a certain platform configuration.
4. **Integrity reporting protocol**	between the platform and the service. This protocol is used for the remote platform integrity reporting.

Protocol **1**, **2**, and **3** is related to the exchange of a TPM identity certificate and a validation data certificate. This specification does not define protocols for exchanging a TPM endorsement certificate, a platform endorsement certificate and a platform conformance certificate because they are usually created and distributed offline.

Protocol **4** and **5** are related to the remote access to TCG functionalities. Since the platform integrity attestation is often performed before other calls, it is defined independently.

In addition, Section 5.7 defines a protocol that cryptographically binds a token to a Trusted Mobile Device. This protocol enables the e-Ticket scenario where the ticket cannot be duplicated by the device user.

5.3 Privacy CA Protocol

Privacy CA protocol is the communication between the platform and the privacy CA to assess a platform and attest to the TPM identity. The resultant TPM identity certificate is an X.509 public key certificate created and signed by the Privacy CA. **[TCG]** defines the message formats for this protocol. These messages have unique syntax since they are designed to preserve the pseudonymity of the identity. For this reason, the TPM identity certificate is never transmitted in plaintext. In this specification, the certificate that is encrypted and encoded is defined as a special binary security token.

5.3.1 Token formats

In this specification, identity certificate request and response are exchanged using Web Services Trust Language (WS-Trust) messages. The WS-Trust specification **[WS-Trust]** defines a set of XML elements that are placed under SOAP body elements and are used to request and issue security tokens. Security token is a representation of a collection of any claims, so what to be defined here are the definitions of token formats for this Privacy-CA protocol.

5.3.1.1 Identity Certificate Request Token

This token MUST be placed in *tmp:TpmIdentityCredentialRequest* element when requesting a TPM identity certificate. The format of the token is defined in Section 4.30.2 and 9.4.1 of **[TCG]**.

5.3.1.2 Identity Certificate Response Token

This token MUST be an encrypted and encoded TPM identity certificate. It is returned from the Privacy CA to the platform. The format of the token is defined in Section 9.4.2 of **[TCG]**.

tmp:TpmIdentityCredential is specified as the type of the security token.

5.3.2 Requesting an Identity Certificate

Below is an example of sending the TPM identity certificate request as a TCG-specific certificate request (using the *ReqIssue* request type **[WS-Trust]**). This example is a SOAP message and all elements other than the identity certificate request token are defined in the WS-Trust spec **[WS-Trust]**.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="req">
 <wst:RequestSecurityToken>
  <wst:TokenType>tmp:TpmIdentityCredential</wst:TokenType>
  <wst:RequestType>wst:ReqIssue</wst:RequestType>
  <tmp:TpmIdentityCredentialRequest>
    ZyBha2FuIGRpcGFrYWkNCkRFU0NSSVBUSU9OPVBpbGloIGJhaGFzYSB1bnR1ayBk
    aXBhc2FuZyBkYXJpIGRhZnRhciBkaWJhd2FoIGluaQ0KT0s9U2V0dWp1DQpDYW5j
    ZWw9UGVtYmF0YWxhbg==
  </tmp:TpmIdentityCredentialRequest>
 </wst:RequestSecurityToken>
</S:Body>
</S:Envelope>
```

5.3.3 Returning an identity Certificate

Below is an example SOAP message of returning a TCG-specific response (encrypted certificate) as a binary security token.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu=http://schemas.xmlsoap.org/ws/2002/07/utility
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/06/secext"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="res">
 <wst:RequestSecurityTokenResponse>
  <wst:TokenType>tmp:TpmIdentityCredential</wst:TokenType>
  <wst:RequestedSecurityToken>
   <wsse:BinarySecurityToken
     ValueType="tmp:TpmIdentityCredential"
     EncodingType="wsse:Base64Binary">
    MIIEZzCCA9CgAwIBAgIQEmtJZc0...
   </wsse:BinarySecurityToken>
  </wst:RequestedSecurityToken>
 </wst:RequestSecurityTokenResponse>
</S:Body>
</S:Envelope>
```

5.4 Validation Data Protocol

The validation data certificate typically represents an assertion by the component validation entity that the component instructions referenced by the certificate have the attributes conveyed in the certificate.

There are two types of validation data request: push mode and pull mode. In push mode, the platform gets the validation data certificate for the current value of platform integrity metrics (a blue paper-like icon in *Figure 5-2 Push Mode Validation Data Protocol*). Then this certificate can be attached in a message to the server.

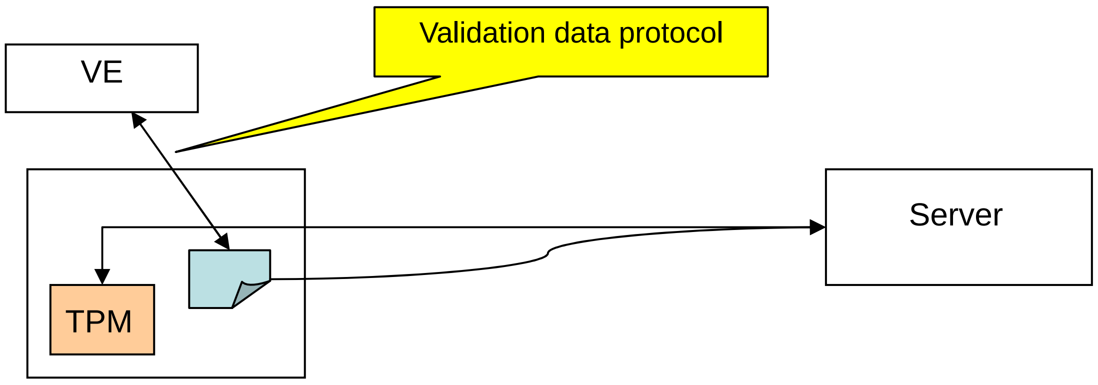


Figure 5-2 Push Mode Validation Data Protocol

On the other hand, in pull mode, the server requests that the Validation Entity (VE) validates the platform configuration.

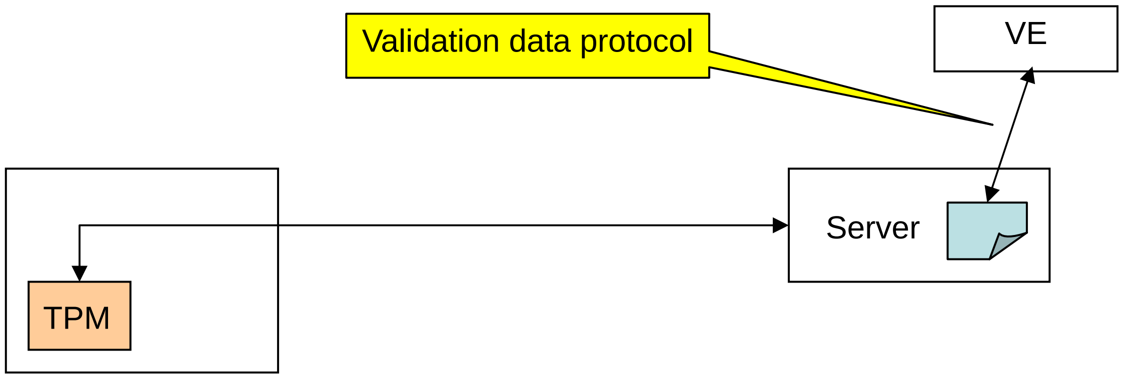


Figure 5-3 Pull Mode Validation Data Protocol

5.4.1 Token formats

5.4.1.1 Validation Certificate Request Token

Validation certificate request is an X.509 attribute certificate request. This is used by the platform to request a validation data certificate to the Validation Entity.

The format of the request: is defined in **[CRMF]**

5.4.1.2 Validation Certificate Response Token

Validation certificate response token is a binary security token that is an X.509 attribute certificate representing the credential for the validation data. The format and the syntax of the validation data certificate is described in Section 4.32.4 and 9.5.4 of **[TCG]**.

In **[TCG]**, the certificate is intended to describe the integrity metric of each component. However, in a real situation, it is often more of interest to consider whether a set of hardware and software components can be trusted as a whole. Therefore, the validation certificate response MAY refer to a set of components. When the certificate refers to such a set of components, the certificate MAY contain the “PCR composite hash” attribute and the “PCR indices” attributes. If these attributed is not specified, the *Holder* field MUST be the value of the *`TCG_COMPOSITE_HASH`* (defined in Section 10.4.5 of **[TCG]**) and the *Subject Alternative Name* field MUST be the value of the PCR indices array with RDNs of the component set endorser.

`tmp:TCGValidationData` is specified as the type of the security token.

5.4.2 Requesting a Validation Data Certificate in Push Mode

Below is the example SOAP message of requesting a security token: Send the TCG validation data request as an attribute certificate request (using the *`ReqIssue`* request type **[WS-Trust]**)

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="req">
 <wst:RequestSecurityToken>
  <wst:TokenType>tmp:TCGValidationData</wst:TokenType>
  <wst:RequestType>wst:ReqIssue</wst:RequestType>
  <tmp:ValidationDataRequest>
    ZyBha2FuIGRpcGFrYWkNCkRFU0NSSVBUSU9OPVBpbGloIGJhaGFzYSB1bnR1ayBk
    aXBhc2FuZyBkYXJpIGRhZnRhciBkaWJhd2FoIGluaQ0KT0s9U2V0dWp1DQpDYW5j
    ZWw9UGVtYmF0YWxhbg==
  </tmp:ValidationDataRequest>
 </wst:RequestSecurityToken>
</S:Body>
</S:Envelope>
```

5.4.3 Returning a Validation Data Certificate in Push Mode

Below is the example SOAP message of returning the validation data attribute as a binary security token

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
```

```
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/06/secext"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="res">
 <wst:RequestSecurityTokenResponse>
  <wst:TokenType>tmp:TCGValidationData</wst:TokenType>
  <wst:RequestedSecurityToken>
   <wsse:BinarySecurityToken
     ValueType="tmp:TCGValidationData"
     EncodingType="wsse:Base64Binary">
    MIIEZzCCA9CgAwIBAgIQEmtJZc0...
   </wsse:BinarySecurityToken>
  </wst:RequestedSecurityToken>
 </wst:RequestSecurityTokenResponse>
</S:Body>
</S:Envelope>
```

5.4.4 Attaching a Validation Data Certificate

Below is the example SOAP message of attaching a validation data certificate in the header of arbitrary SOAP message. This particular example does not specify when and how this message is transmitted. One most typical usage is that the certificate is attached in the message for integrity reporting protocol.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/06/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Header>
 <wsse:Security>
  <wsse:BinarySecurityToken
     ValueType="tmp:TCGValidationData"
     EncodingType="wsse:Base64Binary">
    MIIEZzCCA9CgAwIBAgIQEmtJZc0...
  </wsse:BinarySecurityToken>
 </wsse:Security>
</S:Header>
```

```
<S:Body wsu:Id="res">
  :  <!–- body of the request -->
</S:Body>
</S:Envelope>
```

5.4.5 Requesting a Validation Data Certificate in Pull Mode

Below is an example SOAP message of sending the validation request (*ReqValidate* **[WS-Trust]**), which is represented as an (unsigned) security token. Alternatively, the same protocol as push model can be used.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="req">
 <wst:RequestSecurityToken>
  <wst:TokenType>tmp:TCGValidationData</wst:TokenType>
  <wst:RequestType>wst:ReqValidate</wst:RequestType>
  <tmp:ValidationDataRequest>
    ZyBha2FuIGRpcGFrYWkNCkRFU0NSSVBUSU9OPVBpbGloIGJhaGFzYSB1bnR1ayBk
    aXBhc2FuZyBkYXJpIGRhZnRhciBkaWJhd2FoIGluaQ0KT0s9U2V0dWp1DQpDYW5j
    ZWw9UGVtYmF0YWxhbg==
  </tmp:ValidationDataRequest>
 </wst:RequestSecurityToken>
</S:Body>
</S:Envelope>
```

5.4.6 Returning a Validation Data Certificate in Pull Mode

Below is an example of returning the validation data attribute as a binary security token

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/06/secext"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2002/12/secext"
   xmlns:tmp="http://schemas.trusted-mobile.org/protocol">
<S:Body wsu:Id="res">
  <wst:RequestSecurityTokenResponse>
  <wst:TokenType>tmp:TCGValidationData</wst:TokenType>
  <wst:RequestedSecurityToken>
   <wsse:BinarySecurityToken
     ValueType="tmp:TCGValidationData"
     EncodingType="wsse:Base64Binary">
    MIIEZzCCA9CgAwIBAgIQEmtJZc0...
   </wsse:BinarySecurityToken>
  </wst:RequestedSecurityToken>
 </wst:RequestSecurityTokenResponse>
</S:Body>
</S:Envelope>
```

5.5 Integrity Reporting Protocol

The PCR registers contain the integrity metrics of the platform. These values can be reported by using *TPM_Quote*, the quote function defined in Section 6.3.3 of **[TCG]**. When this function is called, the TPM collects the values of current Platform Configuration Registers (PCRs), combines them with a nonce to generate a special structure called *TCG_QUOTE_INFO* (defined in Section 4.29 of **[TCG]**) and return the signature on this structure. There are two scenarios where attestation signature (signature created by *TPM_Quote*) is useful:

1. **Online Attestation:** A service may want to check the integrity of the platform it is communicating with. In this case, the freshness of the attestation signature must be verified; i.e., it must be bound to a fresh nonce (Figure 5-4). In this specification, online attestation can be performed in two ways. One is to use the *RSA_PCR* signature algorithm in SSL/TLS (see Section 5.5.1.4). The other is to use the *RSA_PCR* signature algorithm in the *Context Establishment protocol* (4.7).

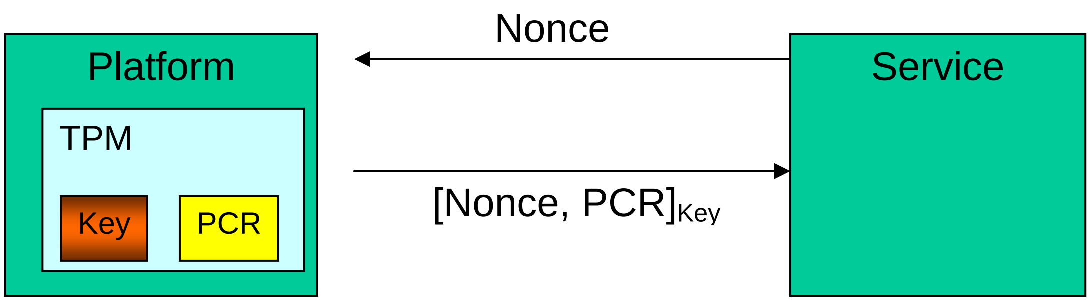


Figure 5-4 Online Attestation

2. **Content attestation:** A platform may want to have a proof of its integrity at the time of signing a document. In this case, content must be bound to the PCR values at the time of signing. The verifier (possibly the service) knows that the signature on the document is done by a platform of a known configuration (See Figure 5-5)

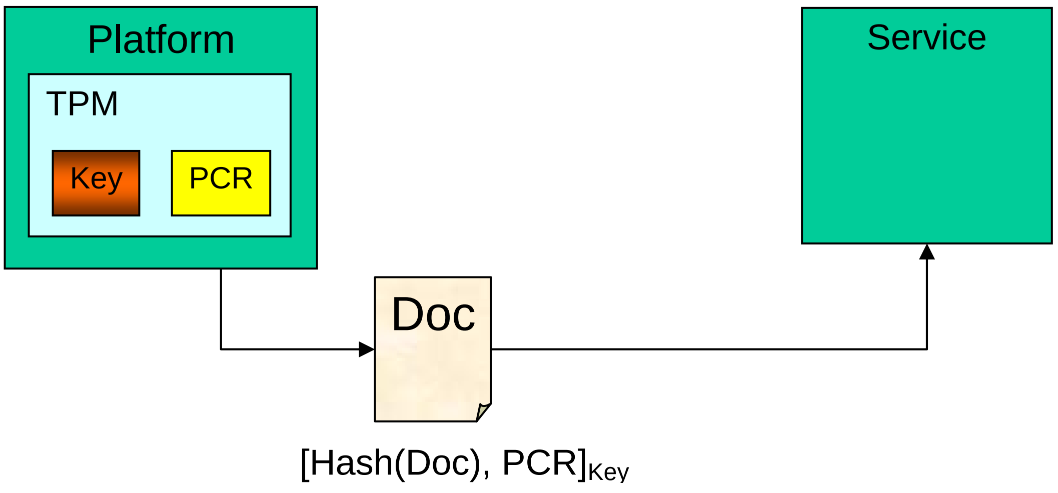


Figure 5-5 Content Attestation

To enable both scenarios, this specification defines a new signature algorithm format that can be used in various existing protocols including SSL/TLS and XML Signature.

5.5.1 Signature algorithm with integrity metrics

A set of two operations, *Sign_RSA_PCR* and *Verify_RSA_PCR*, constitutes a new signing / verification algorithm based on *TPM_Quote*. Specifically, the signature value of *Sign_RSA_PCR* contains the PCR value information at the time of signing. Since *Sign_RSA_PCR* uses *TPM_Quote*, the key length is always 2048 bits.

5.5.1.1 Sign_RSA_PCR

Sign_RSA_PCR takes three inputs: the first value is a specification of the key to be used. The second input a 160 bit data containing an arbitrary data to be signed, which is passed as the externalData parameter in the *TPM_Quote* call. The third input is a specification of PCR registers whose values are to be included in the signature.

The signature, which is the output of *Sign_RSA_PCR*, is defined as a concatenation of *TCG_QUOTE_INFO* as defined in Section 4.29 of **[TCG]** and the value of *sig* returned by the *TPM_Quote* call. Since *TCG_QUOTE_INFO* is always 48 bytes and *sig* is always 256 bytes, the total length of the signature is always 304 bytes.

5.5.1.2 Verify_RSA_PCR

Verify_RSA_PCR takes one input parameter: the signature created by *Sign_RSA_PCR*. In the verification process, the signature is first split into two parts; the first 48 bytes of the signature represent *TCG_QUOTE_INFO* and the rest part is the value of the RSA signature done by the *TPM_Quote* call. Then the following things MUST be tested:

The first part contains a valid *TCG_QUOTE_INFO* structure. This means the version field MUST be the correct, and the fixed field MUST be the ASCII string “QUOT”.

The `externalData` field of the first part is the expected value (i.e., either the nonce or the hash of the signed document).

The second part MUST be the RSASSA-PKCS1-v1.5 signature of the first part according to the given public key (of the TPM identity key.)

5.5.1.3 Bindings of Sign_RSA_PCR and Verify_RSA_PCR

`Sign_RSA_PCR` and *`Verify_RSA_PCR`* can be used anywhere this signature is useful. When they are applied in an existing protocol, the way to specify this signature algorithm is required. In this specification, we define two such bindings: SSL/TLS binding and XML Signature binding. These bindings are RECOMMEND ways to use this attestation signature algorithm.

5.5.1.4 SSL/TLS Binding

The SSL/TLS Binding is useful when online attestation is necessary. In SSL/TLS **[TLS]**, the signature algorithm used in the Handshake Protocol is specified as a CipherSuite. In this specification, we define a single CipherSuite, *`TLS_RSA_PCR_WITH_3DES_EDE_CBC_SHA`* (this needs to be submitted to ITEF WG and be approved). This CipherSuite indicates that *`Sign_RSA_PCR`* and *`Verify_RSA_PCR`* MUST be used when performing the client authentication. The server side authentication is the normal RSA operation as defined in **[TLS]**.

5.5.1.5 XML Signature Binding

The XML Signature Binding is useful when content attestation is necessary. In XML Signature, the algorithm is specified by an algorithm URI. This specification defines the following URI for *`Sign_RSA_PCR`* and *`Verify_RSA_PCR`*.

http://www.trustedcomputinggroup.org/algorithms/xmlsig-pcr

5.6 Schema Definition

The schema for this specification is described below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tmp="http://schemas.trusted-mobile.org/protocol"
  targetNamespace="http://schemas.trusted-mobile.org/protocol">
  <xsd:element name="TpmIdentityCredentialRequest"
    type="xsd:base64Binary" />
  <xsd:element name="ValidationDataRequest" type="xsd:base64Binary" />

  <xsd:simpleType name="TokenTypeEnum">
    <xsd:restriction base="xsd:QName">
      <xsd:enumeration value="tmp:TpmIdentityCredential" />
      <xsd:enumeration value="tmp:TCGValidationData" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

5.7 Remote Data Binding

5.7.1 Definition of Key Remote Data Binding

Often in the physical world we carry objects that give us permissions, but we do not have the ability or authority to modify or duplicate them. In some cases, we do not even have the ability to examine the contents of these objects, or are not aware of (nor care) about the contents. Examples include the SIM cards used in our cell-phones, or the magnetic stripe on subway tickets. These objects act as tickets that grant us access, in one case to a cell-phone network, in the other case to a subway.

In the digital world, it is convenient to be able to construct these types of objects for use. And, in fact, several cryptographic techniques have been used to create non-forgeable tokens. What has not been accomplished is the ability to bind a token to a person (or device) with that person's consent, but without that person being able to modify or duplicate the token. This type of functionality is necessary in cases where duplication would be harmful. An electronic ticket is one such example.

Remote Data Binding uses features of a TPM to bind a token (e.g. a ticket) to a device with the user's consent. However, following this binding, the user is not able to modify or duplicate the token.

5.7.2 Remote Data Binding Protocol

Figure 5-6 depicts the remote data binding protocol for an electronic ticket. The paragraphs following describe the elements of this protocol.

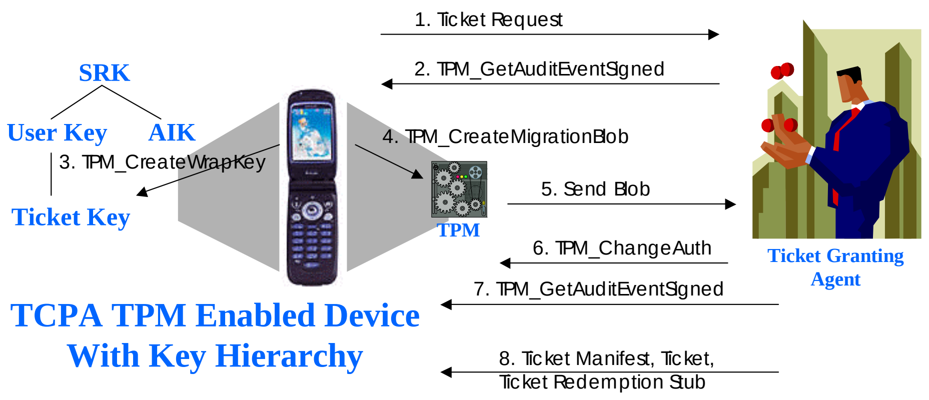

1. Ticket Request
2. TPM_GetAuditEventSigned
SRK
User Key
AIK
3. TPM_CreateWrapKey
Ticket Key
4. TPM_CreateMigrationBlob
TPM
5. Send Blob
6. TPM_ChangeAuth
7. TPM_GetAuditEventSigned
Ticket Granting Agent
TCPA TPM Enabled Device With Key Hierarchy
8. Ticket Manifest, Ticket, Ticket Redemption Stub

Figure 5-6 Remote Data Binding Protocol

The TPM is embedded in a mobile device, and the user of the device has access to the Storage Root Key (SRK), at least one Attestation Identity Key (AIK), and a personal storage key (Users Key). The remote data binding protocol will create a ticket key under the protection of the TPM and provide a ticket to the device protected under that key.

Once the ticket request is provided to the ticket granting agent (**step 1**), the user's device will perform ordinary TPM commands to create a migratable key inside the TPM (**step 3**). The key is extracted securely in a migratable blob (**step 4**), and then shipped to the ticket granting agent (**step 5**). The ticket granting agent then changes the authentication value for the key in a secure manner (**step 6**). In order to ensure that the key is not tampered with before the authentication value is changed, the ticket granting agent performs two audits surrounding the other protocol steps (**step 2** and **step 7**). This allows the ticket granting agent to check the actions that were performed on the TPM and the migratable key, ensuring that the user did not also migrate the key to anyone else (including themselves).

Once this process is completed, the ticket granting agent can ensure that the key in the TPM was properly created, and only the ticket granting agent has access to the key itself. These properties are guaranteed by the TPM. The ticket granting agent can then use this key to protect data and bind that data to the TPM's device.

5.7.3 Ticket Creation

The ticket granting agent creates a ticket using the key it received from the TPM through migration. The ticket must be bound to the TPM's device, and the ticket must be protected from modification and duplication. The ticket is provided in three parts: the ticket manifest, the ticket itself, and the ticket redemption stub.

The ticket manifest is a non-redeemable ticket description, signed by the ticket granting agent, which describes what the actual ticket contains. This manifest is required since the ticket itself is encrypted and not able to be inspected by the user. The manifest is signed in order to represent a legal contract that the actual ticket is equivalent to that which is represented in the manifest, and also to protect the manifest from unauthorized modification.

The ticket itself is signed by the ticket granting agent and also encrypted using the TPM key that was migrated to the ticket granting agent in **Step 5** of the protocol.

The ticket redemption stub is used to authenticate the source of the TPM, and contains the authentication value for the ticket key stored in the TPM, and a ticket identifier. It also contains an AIK certificate that will be used to authenticate the TPM prior to redemption of the ticket. The ticket redemption stub is encrypted with the public key of ticket redemption agent.

5.7.4 Remote Data Redemption Protocol

The user begins the ticket redemption process by sending the ticket redemption stub and the ticket manifest to the ticket redemption agent (**step 1**). The manifest acts as a ticket claim by the user of the device, claiming that he holds the ticket in the manifest. The ticket redemption agent determines if this is the appropriate location and date to redeem the ticket represented by the manifest. If the redemption agent approves the ticket manifest, the agent must then authenticate the TPM. Authentication proceeds by requesting a *TPM_Quote* using a nonce value (**step 2**). The TPM produces the requested quote (**step 3**) and sends the quote information to the ticket redemption agent (**step 4**).

If the agent believes the TPM is authentic based on checking the quote data, and the AIK used to sign the quote data is the same AIK that was provided in the ticket redemption stub, then this proves the TMP device that was issued the ticket is conversing with the redemption agent. The agent then constructs a proper *TPM_Unbind* request using the authentication value retrieved from the decrypted ticket stub (**step 5**). The user decrypts the key using the authentication on the parent key (**step 6**). This step authenticates that the user is redeeming the key to the TPM. The TPM decrypts the ticket and returns the decrypted ticket to the user. The user sends the ticket to the ticket redemption agent for redemption (**step 7**). Upon receipt of the decrypted ticket, the agent ensures that the ticket matches the ticket manifest (e.g. using a ticket identifier) that was presented on the original request (in **step 1**). If these match, the ticket redemption is granted.

When the redemption agent provides the *TPM_Unbind* request to the TPM device, the ticket is considered redeemed and not valid for further use. This is because the ticket can be decrypted by the user once the *TPM_Unbind* request is issued. If the user decrypts the ticket and does not return the ticket to the ticket redemption agent, it is possible to duplicate the ticket (because it is now decrypted). This requirement to redeem the ticket at this point is necessary to prevent an early redemption attempt that does not return the ticket to the redemption agent, and then duplicates the ticket. If an duplication attack is discovered, it can be shown that the user had to collude in such an attempt since the both the AIK used for the *TPM_Quote* and the user key that is the parent of the ticket key had to be accessed. Access to both these keys requires an authentication value that only the user knows (see TCG specification for *TPM_Quote* and *TPM_LoadKey*).

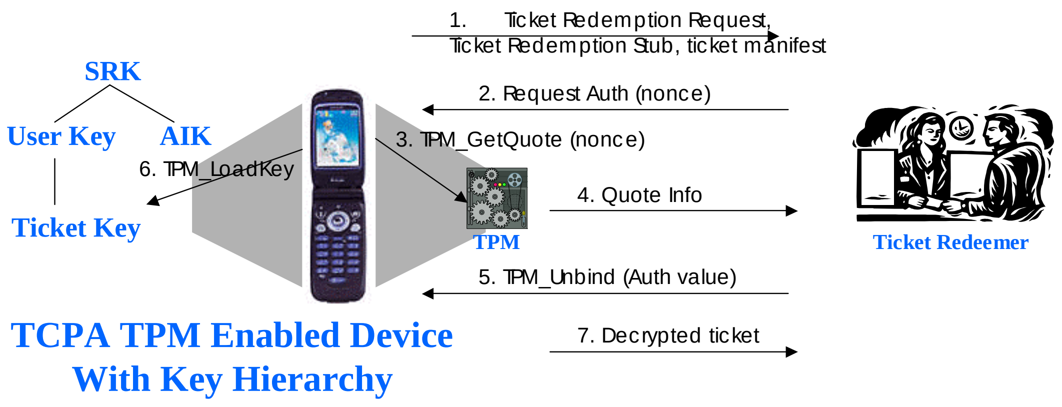


Figure 5-7 Remote Ticket Redemption Protocol

5.8 Security Considerations

This section did not provide explicit protection of the messages exchanged between the device and trusted third parties except for the protection mechanisms that are built-in into the TCG data format. In order to protect the messages in transmission, appropriate mechanisms need to be applied. As discussed in *Section 4*, SSL/TLS with mutual authentication can provide adequate protection if there is a direct TCP/IP connection between the device and the trusted third party. Otherwise, the protection mechanism described in *Section 4* should be employed.

Implementers of this specification should be aware that security tokens (credentials and certificates) must have an expiration time and also they are subject to revocation. In particular, if the push mode of operation (*Section 5.4.3)* is used, the verifier who receives a validation security token should check the status of security tokens using the *`wsse:ReqValidate`* message of WS-Trust according to the platform's policy.

6 Fair Contract Signing Protocol

Reliable and atomic transaction is one of the key requirements for TMP. In the e-Ticket scenario described in the Phase 0 document, when the user purchases an electronic ticket using a mobile device, the paying event and the ticket receiving event must be atomic and inseparable; either they both occur or neither one occurs. Although this is a general requirement for virtually any network-based transactions, TMP has identified that this is a much more serious problem for mobile devices because wireless network connections are less reliable especially when the clients are moving.

This section defines the TMP Fair Contract Signing Protocol, which allows exchanging signatures in an atomic way. The protocol uses the idea of Optimistic Fair Contract Signing recently proposed by Asokan, Shoup, and Waidner **[Asokan1998]**.

6.1 Abstract Protocol Definition

Let us assume party A and party B are exchanging a contract, and T is a trusted third party. *M* denotes the body of the contract. T is always trusted and eventually reachable even if there are network problems. In addition, we assume that A, B, and T can be mutually authenticated.

The basic idea is to first exchange *pre-contracts*, that is, commitments to sign the contract. Once Alice receives Bob's pre-contract in return for her pre-contract, she knows that Bob has committed to sign the contract so she can sign the contract without risk. If Bob refuses to sign the contract, Alice can go to Trent, the trusted third party, with the two pre-contacts (commitments) and have Trent certify the contract.

A pre-contract is defined as a signature on *m*, *A*, and *B*.

A's pre-contract = $sig_A(m, A, B, T)$
B's pre-contract = $sig_B(m, A, B, T)$

Next, we need to define the concept of a *valid contract*.

6.1.1 Two Forms of Valid Contracts

There are two types of valid contract.

6.1.1.1 Standard contract

This is the contract body signed by both parties, that is, a pair of signatures on *m*.

Standard contract = $\{ sig_A(m, A, B), sig_B(m, A, B) \}$

If no messages are lost during the protocol and both parties behave as expected, this is the contract shared by A and B. This type of valid contract can be created without the help of a trusted third party.

6.1.1.2 Notarized contract

If something goes wrong in the protocol, a party who has both pre-contracts can ask *T* for arbitration. Under certain conditions, *T* can issue an alternate form of a valid contract by signing the two pre-contracts as shown below.

Notarized contract = $sig_T(sig_A(m, A, B, T), sig_B(m, A, B, T))$

A notarized contract is created by *T* if either *A* or *B* requests *T* to do so when there is something wrong. By definition, it has the exact same validity as the corresponding standard contract.

The concept of a notarized contract is essential to optimistic fair contract signing. TMP platforms MUST support both forms as valid signatures on message *m*.

6.1.2 Description of Optimistic Fair Contract Signing Protocol

Figure 6-1 illustrates the normal flow of the protocol. The general idea is that *A* and *B* first exchange pre-contracts, then exchange valid contracts.

1. *A* and *B* exchange pre-contracts (commitment to sign the contract)
2. If 1 succeeds, *A* and *B* exchange the actual contract.
3. If 2 also succeeds, the protocol terminates normally. In this case, T is never involved in the transaction.
4. If anything goes wrong, A or B can request that T abort or resolve the transaction. T resolves the transaction by converting a pair of pre-contracts to a valid contract by signing the pair of pre-contracts.

As soon as B receives A's signature in me_3, B can create his own signature and have a standard valid contract. Therefore, the time of agreement formation for B is at the reception of message me_3. At this point, B can be sure that the contract cannot be reversed so he can proceed to the next step of his application.

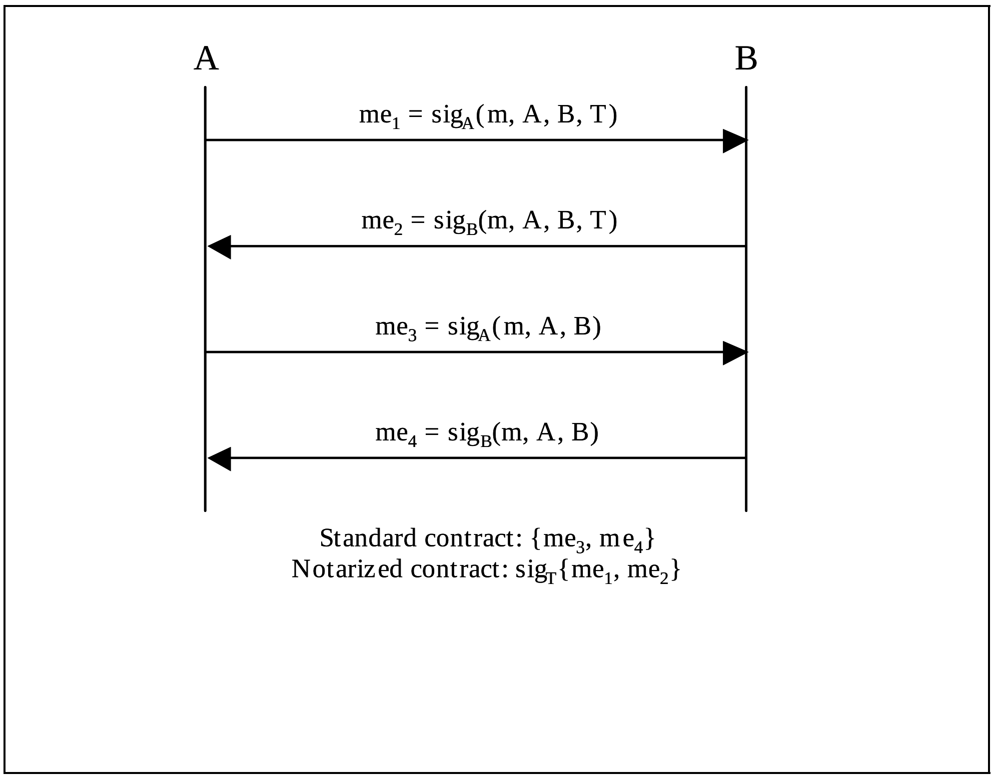

A
B
$me_1 = sig_A(m, A, B, T)$
$me_2 = sig_B(m, A, B, T)$
$me_3 = sig_A(m, A, B)$
$me_4 = sig_B(m, A, B)$
Standard contract: $\{me_3, me_4\}$
Notarized contract: $sig_T\{me_1, me_2\}$

Figure 6-1 Overview of the Optimistic Fair Exchange

If *A* sends her pre-contract (me$_1$) to *B,* but does not receive B's pre-contract (me$_2$), *A* MAY ask *T* to abort the protocol by sending an **abort** message as shown in Figure 6-2. *T* decides if the contract has already been agreed upon and if so, returns a notarized contract to A. If a contract has not been agreed upon, *T* records the abort of the contract and confirms it by returning a signed abort message to A.

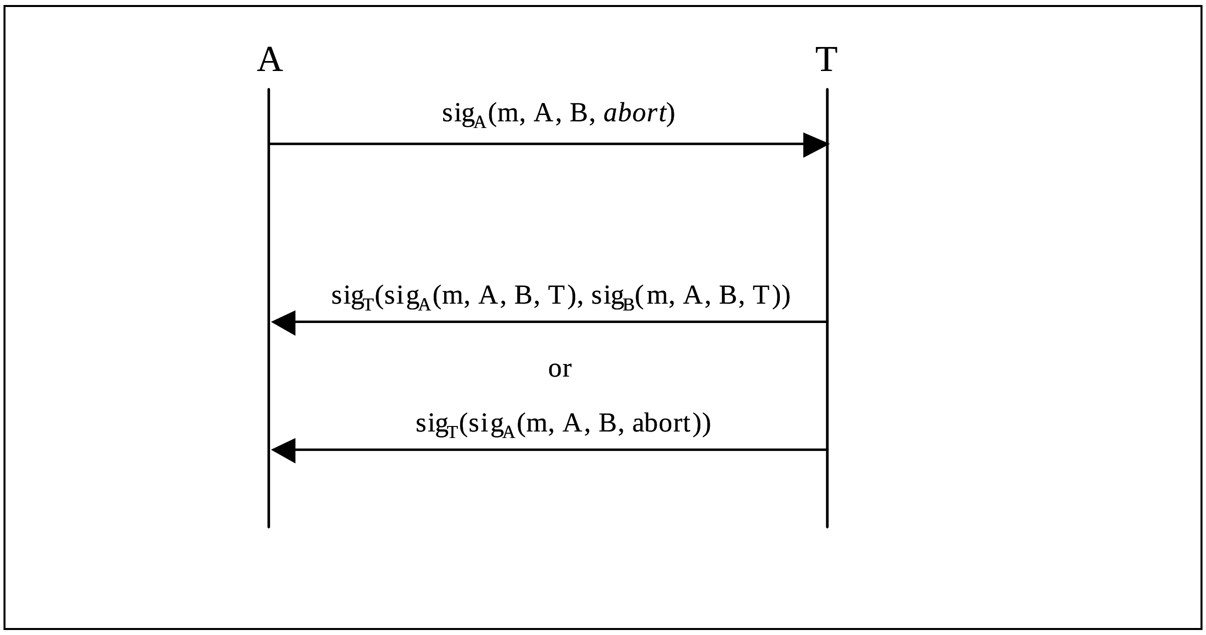

A
T
sig$_A$(m, A, B, *abort*)
sig$_T$(sig$_A$(m, A, B, T), sig$_B$(m, A, B, T))
or
sig$_T$(sig$_A$(m, A, B, abort))

Figure 6-2 Abort Protocol

On the other hand, if *B* sends his pre-contract (me$_2$) but does not receive *A*'s signature (me$_3$), *B* MAY ask *T* to resolve the situation by sending a **resolve** message containing the two pre-contracts to T (Figure 6.3). *T* MUST return a signed **abort** message if the protocol has already been aborted. Otherwise, *T* MUST return a notarized contract.

Likewise, if *A* sends her signature (me$_3$) but does not receive *B*'s signature (me$_4$), *A* MAY ask *T* for a notarized contract using the **resolve** protocol.

Note that the **resolve** protocol and the **abort** protocol MUST be serialized at T so that only one of them is successful, never both.

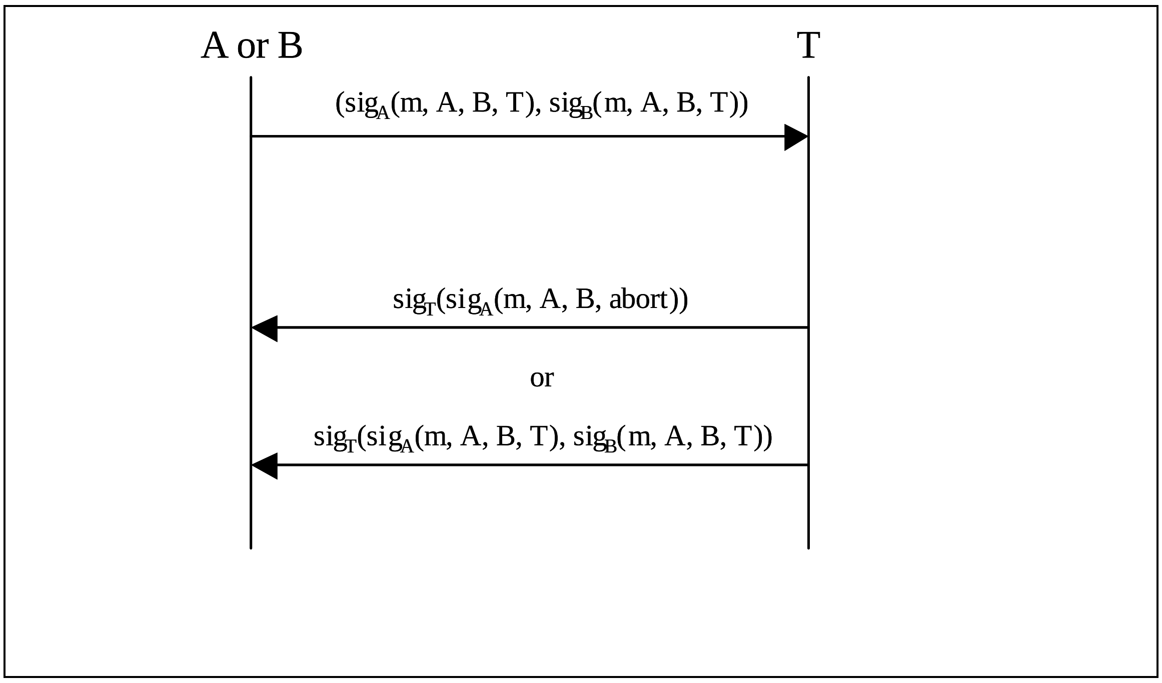

A or B
T
$(sig_A(m, A, B, T), sig_B(m, A, B, T))$
$sig_T(sig_A(m, A, B, abort))$
or
$sig_T(sig_A(m, A, B, T), sig_B(m, A, B, T))$

Figure 6-3 Resolve Protocol

Note also that only *A* can abort the protocol. This is reasonable because it is *A* who makes a commitment (by sending her pre-contract) first. This does not cause any unfairness. Because B cannot abort the protocol, *A* can be sure that the contract is agreed on the receipt of *B*'s pre-contract (me_2). From *A*'s perspective, the time of agreement formation is when it receives me_2 (as long as *A* intends to sign the contract). It is a safe decision because *A* already has the two pre-contracts with which *A* can always obtain a notarized contract from *T.*

6.2 Message Structures

6.2.1 Namespaces

The XML namespace URI **[Namespace] [URI]** that MUST be used by implementations of this specification is:

```
    http://schemas.trusted-mobile.org/protocol/2003/06/ofcs
```

The following namespaces are used in this section. Unless explicitly stated otherwise, the prefixes in this table are assumed to be bound to the corresponding namespace URI's.

Prefix	**Namespace**	**Defined in**
`ofcs`	http://schemas.trusted-mobile.org/protocol/2003/06/ofcs	This document
`ds`	http://www.w3.org/2000/09/xmldsig#	**[XML-Signature]**
`wsse`	http://schemas.xmlsoap.org/ws/2002/12/secext	**[WS-Security]**
`saml`	urn:oasis:names:tc:SAML:1.0:assertion	**[SAML]**

6.2.2 Assertions

The body of a contract MUST be represented as a pair of the SAML <*saml:Assertion*> elements. We assume that the contents of these assertions are mutually agreed upon (but not yet signed) prior to conducting our optimistic fair exchange protocol. For example, Alice browses a catalogue of electronic tickets and decides which ticket she wishes to buy at what price.

6.2.3 Commitments

Next we define an XML representation of a pre-contract $sig_A(m,A,B,T)$. We introduce a new XML element type <*Commitment*> to represent a pre-contract. A commitment consists of two digest values, one for assertion S1 and one for asset ion S2. A commitment also contains the identity of each participant, including the trusted third party *T*. A participant's identity can be represented as an X.509 certificate, which is referenced by a <*wsse:SecurityTokenReference*> element as defined in WS-Security **[WS-Security]**. This construct also allows very flexible and general ways of referring to various other types of identity tokens. A commitment must be signed by A or B depending on who is making this commitment.

The following represents an overview of the syntax of the <*ofcs:Commitment*> element.

```
<ofcs:Commitment CommitmentID=”C1”>
  <ofcs:Initiator>
    <ds:Reference URI=”...”>
     ...
    </ds:Reference>
    </wsse:SecurityTokenReference>
      ...
    <wsse:SecurityTokenReference>
  </ofcs:Initiator>
  <ofcs:Responder>
    <ds:Reference URI=”..”>
      ...
    </ds:Reference>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
  </ofcs:Responder>
  <ofcs:Broker>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
  </ofcs:Broker>
  <ds:Signature>
   ...
  </ds:Signature>
</ofcs:Commitment>
```

The following describes elements and attributes used in a <*ofcs:Commitment*> element.

/Commitment/Initiator
This mandatory element represents the information on the first participant of the protocol.

/Commitment/Initiator/ds:Reference
This mandatory element refers to the assertion made by the first participant. The content of this element conforms to the Reference element of XML Signature.

/Commitment/Initiator/wsse:SecurityTokenReference
This mandatory element refers to a security token that represents the identity of the first participant, such as an X.509 certificate.

/Commitment/Responder
This mandatory element represents the information on the second participant of the protocol.

/Commitment/Responder/ds:Reference
This mandatory element refers to the assertion made by the second participant. The content of this element conforms to the Reference element of XML Signature.

/Commitment/Responder/wsse:SecurityTokenReference
This mandatory element refers to a security token that represents the identity of the second participant, such as an X.509 certificate.

/Commitment/Broker
This mandatory element represents the information on the trusted third party.

/Commitment/Broker/wsse:SecurityTokenReference/
This mandatory element refers to a security token that represents the identity of the trusted third party, such as an X.509 certificate.

/Commitment/ds:Signature
This mandatory element contains an XML signature on the <Commitment> element either by the first participant or the second participant.

Note that we allow indirection for assertion referrals through <*ds:Reference*> rather than directly embedding the assertions within a commitment. There are several reasons for this design.

First, assertions can be arbitrary complex and it might be inconvenient or even infeasible to embed assertions into a commitment. Second, the communicating parties may want to keep the assertions confidential even from the eyes of the trusted third party. If the assertions are embedded in a commitment, they need to be sent to T whenever the **resolve** protocol is executed. Third, a commitment is included in a notarized contract if the standard contract could not be obtained for some reason. Alice may need to present a notarized contract to another party as a proof that Bob has agreed on assertion S2 but Alice may not want to disclose the details of her assertion, S1. For example, Alice may want to keep her credit card payment information (S1) secret when she presents the purchased ticket (S2) to a ticket gate.

6.2.4 Standard Contract

A contract is represented by a new XML element type <*ofcs:Contract*>. As we described in the previous section, there are two types of contracts; standard contracts and notarized contracts. TMP platforms must treat them as having the exact same semantics even though their internal structure may be different. The syntax of a standard contract consists of the digests of the two assertions, S1 and S2, and identities of A and B. A standard valid contract must be signed by both A and B.   A standard contract is valid if and only if both A’s signature and B’s signature are valid according the validator’s policy.

The following represents an overview of the syntax of the <*ofcs:Contract*> element when it is a standard contract.

```
<ofcs:Contract ContractID=”C1”>
  <ofcs:Initiator>
    <ds:Reference URI=”...”>
     ...
    </ds:Reference>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
  </ofcs:Initiator>
  <ofcs:Responder>
    <ds:Reference URI=”..”>
      ...
    </ds:Reference>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
  </ofcs:Responder>
  <ds:Signature> ... </ds:Signature>
  <ds:Signature> ... </ds:Signature>
</ofcs:Commitment>
```

The following describes elements and attributes used in a `<ofcs:Contract>` element.

`/Contract/Initiator`
This mandatory element represents the information on the first participant of the protocol.

`/Contract/Initiator/ds:Reference`
This mandatory element refers to the assertion S1 made by the first participant. The content of this element conforms to the Reference element of XML Signature.

`/Contract/Initiator/wsse:SecurityTokenReference`
This mandatory element refers to a security token that represents the identity of the first participant, such as an X.509 certificate.

`/Contract/Responder`
This mandatory element represents the information on the second participant of the protocol.

`/Contract/Responder/ds:Reference`
This mandatory element refers to the assertion S2 made by the second participant. The content of this element conforms to the Reference element of XML Signature.

`/Contract/Responder/wsse:SecurityTokenReference`
This mandatory element refers to a security token that represents the identity of the second participant, such as an X.509 certificate.

`/Contract/Broker/wsse:SecurityTokenReference/`
This mandatory element refers to a security token that represents the identity of the trusted third party, such as an X.509 certificate

`/Contract/ds:Signature`
These mandatory elements contain XML signatures on the `<Contract>` element by the first participant and the second participant. The signature scope must cover both the `<Initiator>` element and the `<Responder>` element.

An application can treat a valid contract as an alternative form of A’s digital signature on assertion S1 or B’s digital signature on S2 independent from the other assertion. For example, if a valid contract between Alice and Bob contains Alice’s payment assertion S1, a payment application can use this contract in lieu of Alice’s signed payment after verifying that the digest value of S1 is correct.

6.2.5 Notarized Contract

A notarized contract is an alternate form of <*Contract*>. It consists of the digests of two assertions, identities of the two parties, references to two commitments, and T’s signature on them. A notarized contract is valid if and only if the signature is valid. An application that receives a notarized contract needs to verify only one signature instead of two signatures (which is the case for a standard contract). The application does not need to retrieve commitments to check their contents because it can trust *T* to have made the necessary checks when *T* signed the notarized contract.

The following represents an overview of the syntax of the *<ofcs:Contract>* element when it is a notarized contract.

```
<ofcs:Contract ContractID=”C1”>
  <ofcs:Initiator>
    <ds:Reference URI=”...”>
     ...
    </ds:Reference>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
    <ofcs:CommitmentRef RefID=”...”/>
  </ofcs:Initiator>
  <ofcs:Responder>
    <ds:Reference URI=”..”>
      ...
    </ds:Reference>
    <wsse:SecurityTokenReference>
      ...
    </wsse:SecurityTokenReference>
    <ofcs:CommitmentRef RefID=”...”/>
  </ofcs:Responder>
  <ds:Signature> ... </ds:Signature>
</ofcs:Contract>
```

The following describes elements and attributes used in a *<ofcs:Contract>* element.

/Contract/Initiator
This mandatory element represents the information on the first participant of the protocol.

/Contract/Initiator/ds:Reference
This mandatory element refers to the assertion S1 made by the first participant. The content of this element conforms to the Reference element of XML Signature.

/Contract/Initiator/wsse:SecurityTokenReference
This mandatory element refers to a security token that represents the identity of the first participant, such as an X.509 certificate.

/Contract/Initiator/CommitmentRef
This mandatory element refers to the `<ofcs:Commitment>` document by the first participant.

`/Contract/Responder`
This mandatory element represents the information on the second participant of the protocol.

`/Contract/Responder/ds:Reference`
This mandatory element refers to the assertion S2 made by the second participant. The content of this element conforms to the Reference element of XML Signature.

`/Contract/Responder/wsse:SecurityTokenReference`
This mandatory element refers to a security token that represents the identity of the second participant, such as an X.509 certificate.

`/Contract/Initiator/CommitmentRef`
This mandatory element refers to the *`<ofcs:Commitment>`* document by the second participant.

`/Contract/Broker/wsse:SecurityTokenReference/`
This mandatory element refers to a security token that represents the identity of the trusted third party, such as an X.509 certificate

`/Contract/ds:Signature`
These mandatory elements contain an XML signature by the trusted third party on the *`<Contract>`* element.

6.2.6 Aborted Contract

When *A* requests to abort the protocol and this request is confirmed by *T*, *A* should receive a confirmation of the abort as follows.

The following represents an overview of the syntax of the *`<ofcs:AbortedContract>`* element when it is a notarized contract.

```
<ofcs:AbortedContract ContractID=”C1”>
  <ofcs:CommitmentRef RefID=”...”/>
  <ds:Signature> ... </ds:Signature>
</ofcs:AbortedContract>
```

The following describes elements and attributes used in a *`<ofcs:AbortedContract>`* element.

`/AbortedContract//CommitmentRef`
This mandatory element refers to the *`<ofcs:Commitment>`* document by the first participant that has been committed but to be aborted.

`/AbortedContract/ds:Signature`
This mandatory element contains an XML signature by the trusted third party on the *`<AbortedContract>`* element.

6.3 Protocol

Now we define a set of protocols. Following the Web Services protocol tradition, we use <*XXRequest*> and <*XXResponse*> combinations to describe request-response pairs. The following illustrates the basic structure of these messages.

6.3.1 The commitment request from Alice to Bob:

The first round trip between Alice and Bob is an exchange of pre-contracts (commitments). Alice first creates a <*ofcs:commitment*> structure (6.2.3) by signing the assertions and sends an <*ofcs:CommitmentRequest*> message to Bob as follows. This element contains a single element that MUST be <*ofcs:Commitment*>.

```
<ofcs:CommitmentRequest>
  <ofcs:Commitment CommitmentID="C1">
   Signed by Alice ...
  </ofcs:Commitment>
</ofcs:CommitmentRequest>
```

If this message does not conform to this syntax, the signature by Alice does not verify, or Bob does not trust the signing key, Bob MUST abort the protocol and MAY return a SOAP fault as described in Section 4.8.

6.3.2 The commitment response from Bob to Alice

Bob signs the same commitment and returns it to Alice. This message has an <*ofcs:CommitmentResponse*> element whose only child element is the commitment signed by Bob.

```
<ofcs:CommitmentResponse>
  <ofcs:Commitment CommitmentID="C2">
   Signed by Bob ...
  </ofcs:Commitment>
</ofcs:CommitmentResponse>
```

If this message does not arrive before a deadline that is set by Alice's policy, the message does not conform to this syntax, the signature by Bob does not verify, or Alice does not trust the signing key, Alice MUST abort the protocol and MUST initiate the **abort** protocol with Trent (6.3.5).

6.3.3 Contract request from Alice to Bob:

The second round trip normally is an exchange of the signatures of the standard contract. Alice first creates a standard contract using the <*ofcs:Contract*> structure (6.2.4), signs it, and sends it to Bob using the <*ofcs:ContractRequest*> element whose only child is the <*ofcs:Contract*> element.

```
<ofcs:ContractRequest>
  <ofcs:Contract>
    Signed by Alice ...
  </ofcs:Contract>
</ofcs:ContractRequest>
```

If this message does not conform to this syntax, this message does not arrive before a deadline that is set by Bob's policy, the signature by Alice does not verify, or Bob does not trust the signing key, Bob MUST abort the protocol and MUST initiate the **resolve** protocol (6.3.7) with Trent. In addition, Bob MAY send a SOAP fault as described in Section 4.8 to Alice.

6.3.4 Contract response from Bob to Alice:

Bob signs the same contract using his signature key and returns the signed contract, represented as <*ofcs:Contract*> with Bob's signature, to Alice using the <*ofcs:ContractResponse*> element as follows.

```
<ofcs:ContractResponse>
  <ofcs:Contract>
    Signed by Bob ...
  </ofcs:Contract>
</ofcs:ContractResponse>
```

If this message does not conform to this syntax, this message does not arrive before a deadline that is set by Alice's policy, the signature by Bob does not verify, or Alice does not trust the signing key, Alice MUST initiate the **resolve** protocol (6.3.7) with Trent. In addition, Alice MAY send a SOAP fault as described in Section 4.8 to Bob.

6.3.5 Abort request from Alice to Trent:

If Alice does not receive Bob's proper commitment, she needs to abort the transaction by sending an **abort** request to the trusted third party, Trent. This message has a single element, <*ofcs:AbortRequest*>, whose only child element is the commitment that has been sent to Bob.

```
<ofcs:AbortRequest>
  <ofcs:Commitment CommitmentIDRef="C1">
    Signed by Alice
  </ofcs:Commitment>
</ofcs:AbortRequest>
```

If this message does not conform to this syntax, the signature by Alice does not verify, or Trent does not trust the signing key, Trent MUST return a SOAP fault as described in Section 4.8 to Alice.

6.3.6 Abort response from Trent to Alice:

Trent returns either a notarized contract or an aborted contract, depending on the status of the contract. This message consists of an *<ofcs:AbortResponse>*, whose only child element is either *<ofcs:Contract>* that is a notarized contract by Trent as follows

```
<ofcs:AbortResponse>
  <ofcs:Contract contractID="123456789">
    Notarized contract Signed by Trent ...
  </ofcs:Contract>
</ofcs:AbortResponse>
```

or an *<ofcs:AbortedContract>* signed by Trent as follows.

```
<ofcs:AbortResponse>
  <ofcs:AbortedContract contractID="123456789">
    Signed by Trent ...
  </ofcs:AbortedContract>
</ofcs:AbortResponse>
```

If this message does not conform to the above syntax, this message does not arrive before a deadline that is set by Alice's policy, the signature by Trent does not verify, or Alice does not trust the signing key, Alice MUST retry the request until it succeeds.

6.3.7 Resolve request from Alice or Bob to Trent:

Any party who has two commitments can ask Trent to resolve the contract by sending these two commitments. This message consists of a single *<ofcs:ResolveRequest>* element that has two child elements both of which are *<ofcs:Commitment>*. One is Alice's commitment and the other is Bob's commitment.

```
<ofcs:ResolveRequest>
  <ofcs:Commitment> Signed by Alice </ofcs:Commitment>
  <ofcs:Commitment> Signed by Bob </ofcs:Commitment>
</ofcs:ResolveRequest>
```

If this message does not conform to the above syntax, the signature by Alice or Bob does not verify, or Trent does not trust the signing keys, Trent MUST return a SOAP fault as described in Section 4.8 to the requester.

6.3.8 Resolve response from Trent to Alice or Bob:

When Trent receives a <*ResolveRequest*> message, Trent first checks if the contract has already been aborted. If so, he returns an <*AbortedContract*> message. If not, he verifies the signatures on the received commitments and then checks if these commitments are for the same contract by comparing the <*Digest*> elements in each commitment. If they match, then Trent records that the contract has been signed, and returns a notarized contract to the requester.

```
<ofcs:ResolveResponse>
  <ofcs:Contract contractID="123456789">
    Notarized contract Signed by Trent ...
  </ofcs:Contract>
</ofcs:ResolveResponse>
```

or

```
<ofcs:ResolveResponse>
  <ofcs:AbortedContract contractID="123456789">
    Signed by Trent ...
  </ofcs:AbortedContract>
</ofcs:ResolveResponse>
```

If this message does not conform to the above syntax, this message does not arrive before a deadline that is set by the requester's policy, the signature by Trent does not verify, or the requester does not trust the signing key, the requester MUST retry the request until it succeeds.

6.4 Scenario Example

Let us consider the e-ticket purchasing scenario using the protocol defined in this section. Alice wants to purchase a baseball ticket, which is represented as a SAML assertion shown below.

Ticket assertion:

```
  <saml:Assertion
    AssertionID="TicketInfo003"
    Issuer="Company"
    IssueInstant="2002-07-10T13:18:00-05:00">
   <ticket:TicketStatement
      Resource="http://GameServer.com/Game123"
      Decision="Permit">
   <saml:Subject>
     <saml:NameIdentifier NameQualifier="myDev/BobNobody"/>
   </saml:Subject>
   <saml:Action>Enter</saml:Action>
   <ticket:GameName>
     San Francisco Giants vs Seattle Mariners
   </ticket:GameName>
    ...
   </ticket:TicketStatement>
  </saml:Assertion>
```

For this assertion to be a valid ticket, it must be digitally signed by the ticket issuer. Here, instead of having the issuer's signature directly on this assertion, we use our optimistic fair contract signing protocol to obtain a valid contract, which can be used as an alternate form of the issuer's signature.

Alice creates her payment assertion (which authorizes her credit company to pay the ticket price to the ticket issuer) and initiates the protocol. This payment becomes valid if and only if the contract is signed, so she does not need to worry about paying without receiving the ticket.

If no messages are lost and both Alice and Bob are faithful, Alice will receive a valid standard contract as the second return value (i.e., in <*ofcs:ContractResponse*>). In this case, there is no need to contact Trent. She then presents both the ticket assertion and the valid contract at the ticket gate. The ticket gate verifies the contract and checks if the hash value of the ticket matches the hash value in the contract. If they check out, the ticket gate let Alice in. Note that the ticket gate does not need to access Alice's payment information even though it is a part of the signed contract.

If Alice does not receive a standard contract from Bob, Alice still can receive a contract by asking Trent to resolve the contract if she has Bob's pre-contract. In this case, Alice uses a notarized contract instead of a standard contract. The ticket gate should accept both forms of the contract as long as it is valid.

If Alice does not even receive Bob's pre-contract, Alice should contact Trent to see if Bob is willing to sign the contract. Alice will receive either a notarized contract or an <*AbortedContract*> message.

From Bob's point of view, as soon as he receives Alice's signature on the contract, he can conclude that the contract has been signed and can immediately contact the financial institute for the payment resolution by presenting the contract together with Alice's payment statement.

6.5 Schema

An informative graphical representation of the schema for the <*ofcs:Contract*> element is shown in Figure 6-4.

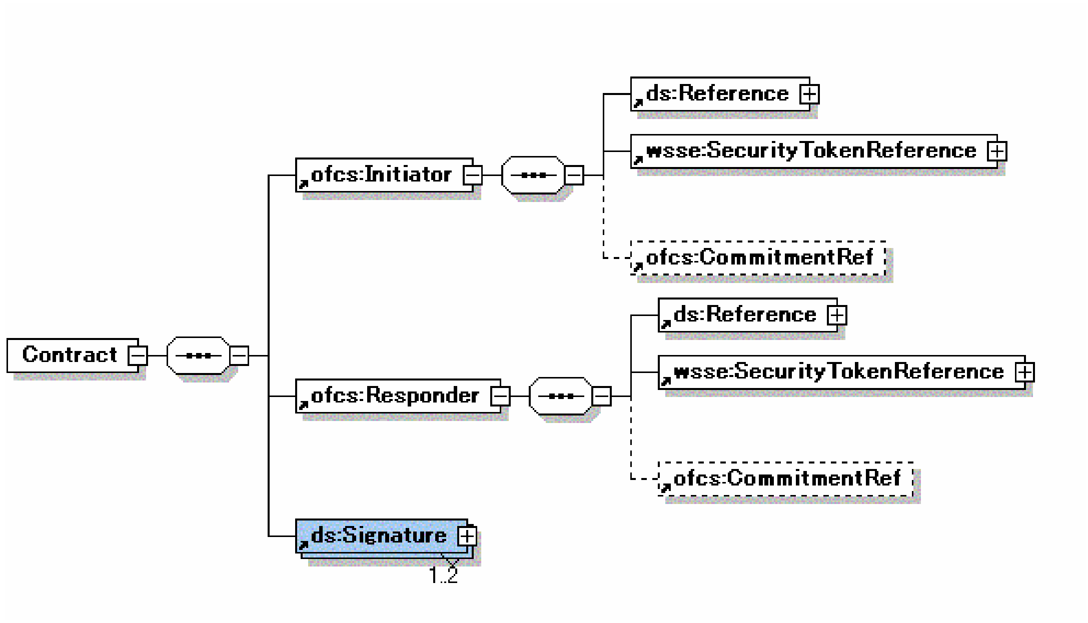


Figure 6-4 Schema Overview of <ofcs:Contract> (informative)

An informative graphical representation of the schema for the <*ofcs:Contract*> element is shown in Figure 6-5.

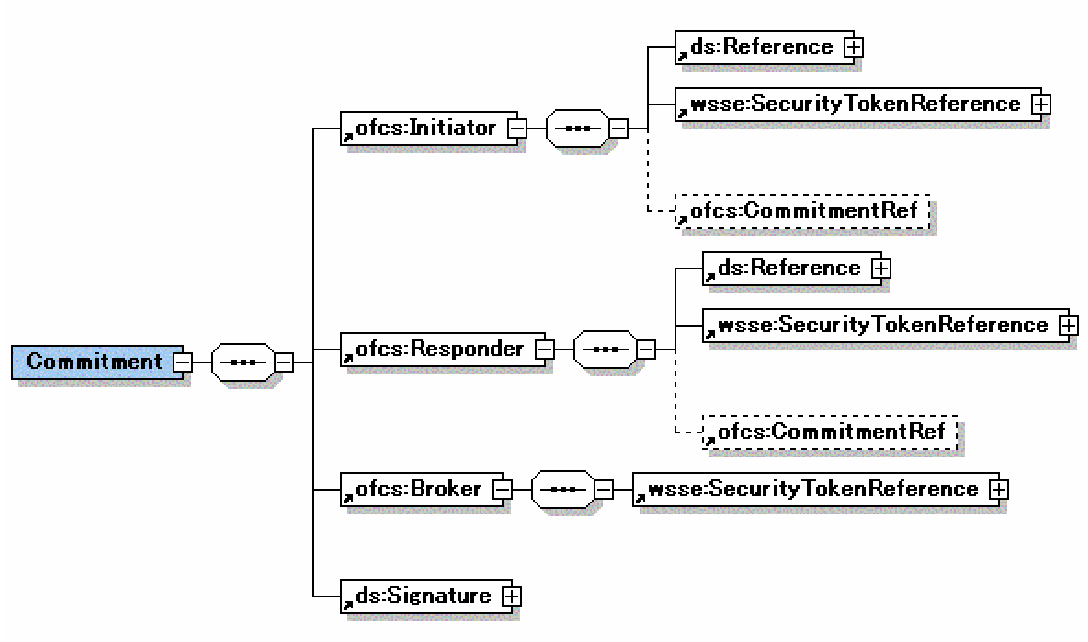


Figure 6-5. Schema Overview of <ofcs:Commitment> (informative)

The complete schema definition for this section is given below.

```
<xsd:schema
targetNamespace="http://schemas.trusted-mobile.org/protocol/2006/06/of
```

```
cs"
xmlns:ofcs="http://schemas.trusted-mobile.org/protocol/2006/06/ofcs"
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:import namespace="http://schemas.xmlsoap.org/ws/2002/12/secext"
schemaLocation="wss.xsd"/>
  <xsd:element name="Commitment">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="ofcs:Initiator"/>
        <xsd:element ref="ofcs:Responder"/>
        <xsd:element ref="ofcs:Broker"/>
        <xsd:element ref="ds:Signature" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="CommitmentID" type="ofcs:IDType"
use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Contract">
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="ofcs:Initiator"/>
          <xsd:element ref="ofcs:Responder"/>
          <xsd:element ref="ds:Signature" minOccurs="1" maxOccurs="2"/>
        </xsd:sequence>
      <xsd:attribute name="ContractID" type="ofcs:IDType"
use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="AbortedContract">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="ofcs:CommitmentRef"/>
        <xsd:element ref="ds:Signature"/>
      </xsd:sequence>
      <xsd:attribute name="ContractID" type="ofcs:IDType"
use="required"/>
```

```
  </xsd:complexType>
</xsd:element>
<xsd:element name="CommitmentRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Commitment"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CommitmentResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Commitment"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ContractRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Contract"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ContractResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Contract"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="AbortRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Commitment"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:element>
<xsd:element name="AbortResponse">
  <xsd:complexType>
    <xsd:choice>
      <xsd:sequence>
        <xsd:element ref="ofcs:Contract"/>
      </xsd:sequence>
      <xsd:sequence>
        <xsd:element ref="ofcs:AbortedContract"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ResolveRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ofcs:Commitment" minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ResolveResponse">
  <xsd:complexType>
    <xsd:choice>
      <xsd:sequence>
        <xsd:element ref="ofcs:Contract"/>
      </xsd:sequence>
      <xsd:sequence>
        <xsd:element ref="ofcs:AbortedContract"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Initiator">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ds:Reference"/>
      <xsd:element ref="wsse:SecurityTokenReference"/>
      <xsd:element ref="ofcs:CommitmentRef" minOccurs="0"
maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="Responder">
   <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="ds:Reference"/>
      <xsd:element ref="wsse:SecurityTokenReference"/>
         <xsd:element ref="ofcs:CommitmentRef" minOccurs="0"
maxOccurs="1"/>
     </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="Broker">
   <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="wsse:SecurityTokenReference"/>
     </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="CommitmentRef">
   <xsd:complexType>
     <xsd:attribute name="RefID" type="ofcs:IDReferenceType"
use="required"/>
   </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="IDType">
   <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="IDReferenceType">
   <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:schema>
```

7 Mobile Device Management

7.1 Introduction

Mobile device management consists of varying features and functionalities to allow the user and or the service provider to install, update, or modify the software and configurations of a Trusted Mobile Device (TMD). This section discusses the process of the software download, key management and the associated protocols and practices.

7.2 Software Download

7.2.1 Definition of Software Download

The purpose of this section is to describe how software download can be deployed remotely, and to identify requirements imposed upon the TMDs to support these deployments. This section will show guidance to mobile network operators and manufactures of the trusted mobile platforms, for the software deployment in a secure and trusted manner. It is RECOMMENDED that a TMD supports the software download capability. If a TMD supports the software download, it MUST follow the statement in this section.

A TMD should provide mechanisms that allow user interaction to discover new software patches and/or upgrades that can be loaded into the TMD. A download may be triggered by a software provider in the form of a notification (e.g., message push). In either case, this process MUST be implemented in some secure form or fashion. For example, the discovery application may be a browser based or native application, as long as it supports a secure mechanism as described further in this section. Different permutations may exist in facilitating a secure means of negotiation. Solutions for the core software (e.g., OS kernel, browser) download may differ from those for an application software download.

Other means of installation utilizing local or proximity communication mechanisms (e.g. Bluetooth™ wireless technology, serial cable, IrDA™, etc...) MAY also be supported by a TMD, but are outside of the scope of this document. An example of a software download may consist of the following processes: initiation, secure context establishment (including mutual authentication), capability negotiation, download acceptance, software download integrity test, and installation of the software. Additional requirements (and use cases) may be identified if the software downloading capability is extended to support temporary or semi-permanent software.

7.2.2 Software Download Requirements

The software download protocol supports the following functional and security capabilities to allow downloading the core software and highly trusted applications.

Authenticity	Authenticity of the downloaded software MUST be ensured by either one or both of the following methods: 1) The TMD MUST authenticate the software provider before exchanging security sensitive information. The software provider SHOULD authenticate the TMD. 2) The TMD SHOULD authenticate the downloaded software using the attached digital signature.

Integrity	Integrity of the data and messages exchanged between two parties SHOULD be securely protected to avoid forgery. Especially the integrity of the software being downloaded MUST be ensured.
Confidentiality	Data and messages exchanged between two parties MUST be encrypted if the information requires confidentiality.
Full Updates	Full update (i.e., to update whole OS) as well as partial, differential updates SHOULD be supported, in a secure manner.
User Consent	The installation process SHOULD take into account the user's preference to defer the process, since the installation process MAY interrupt the user's current activity on the TMD. The installation could consume the rest of the battery power on the TMD, which SHOULD be avoided if the user is not willing to risk loss of service due to a low battery condition. The TMD user SHOULD be able to make the final decision on whether to install the software, not to install, or postpone is the install until some future time convenient for the user.

7.2.3 Core Software Download Scenario

The core software download may be initiated by a user of a TMD to start downloading from a Trusted Software Configuration Service (TSCS). In this document, a TSCS is defined as a conceptual entity which is responsible for keeping track of versions and configurations of the trusted core software (e.g., OS). In the real world, a TSCS may be provided by an entity such as a network operator, manufacturer, or a delegate of such a trusted entity. The TSCS SHOULD then authenticate the TMD. The TSCS and the TMD MAY then exchange capability information of the TMD, to see whether the TMD is capable of accepting, installing and executing the downloadable software code. If both parties (TMD and TSCS) agree that the download is possible, the TSCS SHOULD send a download acceptance message, which MAY provide the proper information on certification / type approval of the software code, download procedures, scheduling and installation options. After receiving the downloaded software, the TMD SHOULD conduct a software integrity test by verifying the signature on the downloaded software. Upon the TMD passing all the qualifying test suites, the software installation SHOULD takes place at the mobile device user's consent.

7.2.4 Application Software Download Scenario

An application software download takes place in a similar manner as the core software download, except that the software provider is not necessarily a TSCS. Some security requirements may be lightened for downloading less-trusted applications, while only highly trusted applications should be able to access protected capabilities of the TMD.

7.2.5 Software Download Threat Analysis and mitigating method

Software download is one of the many key features residing in the TMD. Considering the high security nature of this feature and its associated protocols, it is pertinent to address the security threats that the software download function MUST mitigate.

Unauthorized Server

Malicious / fictitious server masquerading as a trusted server
(*See 7.2.6.1 – Mutual Authentication*)

Unauthorized Client

Malicious / Competitor trying to access the Server.

(See 7.2.6.1 – *Mutual Authentication*)

Malicious Software

Update Software with malicious intent to harm infrastructure and/or to deviate from the mobile device's expected behavior.

(See 7.2.6.1 – *Mutual Authentication*)

Man-in-the-Middle Attack

An attack where the attacker is able to read, and possibly modify at will, messages between two parties without letting either party know that they have been attacked. The attacker is able to observe and intercept messages going between the two victims.

(See 7.2.6.1 – *Mutual Authentication*)

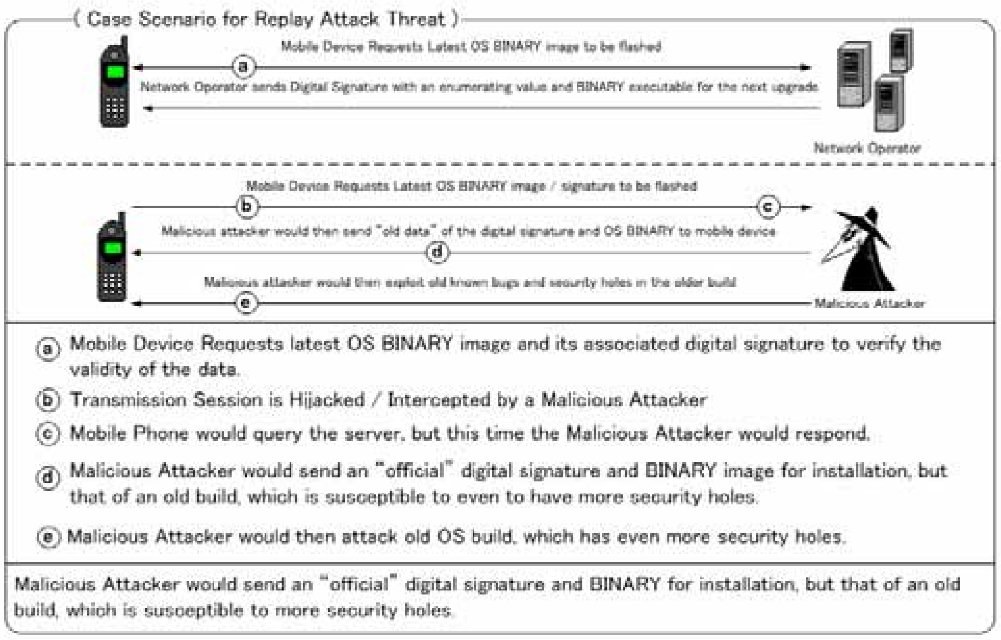
Replay Attack

An attack in which a valid data transmission is maliciously or fraudulently repeated, either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack.

(See 7.2.6.1 – *Mutual Authentication*)

Even though the integrity of the software code to be upgraded is assured by a digital signature, TMDs are still prone to probable threats such as *replay attacks*. *Figure 7-1 Case Scenario for the Replay Attack Threat* illustrates a possible replay attack by a malicious attacker with the intent to circumvent the data transmission between a TSCS and a TMD. The attacker may intercept communication between a TSCS and TMD to obtain a copy of the software code with a valid signature of the TSCS. Occasionally vulnerabilities are found in an old version of software and the new version is released to fix the vulnerability. The malicious attacker can again intercept the communication between the TSCS and a TMD, and masquerade as old vulnerability prone software, as the newer build; by using a replay attack technique. When the TMD credulously installs the old software, the malicious attacker can exploit the older vulnerabilities.

See **[WG2ThreatModel]** for more comprehensive analysis of threat models on Trusted Mobile Platforms.


(Case Scenario for Replay Attack Threat)
Mobile Device Requests Latest OS BINARY image to be flashed
a
Network Operator sends Digital Signature with an enumerating value and BINARY executable for the next upgrade
Network Operator
Mobile Device Requests Latest OS BINARY image / signature to be flashed
b
c
Malicious attacker would then send "old data" of the digital signature and OS BINARY to mobile device
d
Malicious attacker would then exploit old known bugs and security holes in the older build
e
Malicious Attacker

(a) Mobile Device Requests latest OS BINARY image and its associated digital signature to verify the validity of the data.

(b) Transmission Session is Hijacked / Intercepted by a Malicious Attacker

(c) Mobile Phone would query the server, but this time the Malicious Attacker would respond.

(d) Malicious Attacker would send an "official" digital signature and BINARY image for installation, but that of an old build, which is susceptible to even to have more security holes.

(e) Malicious Attacker would then attack old OS build, which has even more security holes.

Malicious Attacker would send an "official" digital signature and BINARY for installation, but that of an old build, which is susceptible to more security holes.

Figure 7-1 Case Scenario for the Replay Attack Threat

7.2 6 Core Software Download Protocols

7.2.6.1 Mutual Authentication

The concept of the mutual authentication is provided in Section 8.3.4 (*Two-Way Authentication*) The client and service SHOULD prove respective identity to each other by exchanging an authorization token or another form of valid credential, before starting a transaction. After authenticating each other, two parties can exchange a common session key that can be used to establish a secure communication context as discussed in Section 4. (*Context Establishment*), SSL/TLS is one example method for mutual authentication and secure communication context. The use of digital signatures should be used to prove integrity and of the data and the identity of the data provider.

7.2.6.2 Software Download Protocol

The software download protocol SHOULD be designed to provide protection against unauthorized disclosure and modification for all elements downloaded onto the TMD.

The software download protocol SHOULD observe the following technologies to achieve secure transaction:

SSL / TLS:

The utilization of SSL/TLS can mitigate the attacks listed above. SSL/TLS server authentication MUST be utilized to authenticate the server identity. In order to prevent an unauthorized client from accessing the server, the Software Download Protocol SHOULD utilizes the SSL/TLS client authentication function or some other means to authenticate the client.

SSL/TLS with TPM extension:

An attestation signature SHOULD be utilized in the SSL/TLS handshake protocol, to prove the identity and integrity of the TMD. An attestation signature may be conveyed using the TLS Extension Mechanism (see Section 7.3.5.2 (*Use of SSL and TLS*) and **[TLSExt]**) or the SSL/TLS binding described in Section 5.5.1.4 (*SSL/TLS Binding*).

Digital Signature (with a timestamp or nonce):

The utilization of digital signatures ensures the integrity of the downloaded software. The software provider SHOULD sign the software code along with a timestamp or nonce, to allow the TMD to detect replay attacks as well as to make sure that the downloaded software is the newest version. A TMD MAY generate a nonce and send it to the software provider before downloading the software, in which case the software provider SHOULD include the nonce in the signature that authenticates the software code being downloaded.

An example data exchange MAY be comprised of two parts, a header and a payload. The header component consists of the software identification and a reference to the software provider. This software identification can also be represented through a digital signature from the software provider. A version number SHOULD be used mainly to identify different software builds. A digital signature SHOULD be used to verify the integrity of the software code.

The payload is rather self explanatory. A payload contains software code that needs to be upgraded, such as a hardware-specific binary image for that designated TMD, or a JAR file for the code written in Java. Optional agents to perform the software installation may be placed in this payload, which also needs to be integrity protected using digital signatures.

7. 3 Key Management

7.3.1 Definition of Key Management

Key management is defined as the attributes in accordance with a security policy referred in **[ISO7498-2, NISTIR-4972, and ISO11770-1].** Further more, [**ISO7498-2]** key management is of Security Management, but does not dictate where this functionality must reside in a system or protocol stack.

7.3.1.1 Trusted Mobile Platform Key Management Protocols

Key management is concerned with more than just preventing keys from being revealed to unauthorized entities. **[ISO11770-1]** specifies that Key management is susceptible to a number of threats. These are:

- Disclosure of key material
- Modification of key material
- Unauthorized Deletion of key material

- Unauthorized Revocation of keys
- Masquerade of authorities or users to gain access to key management services
- Delay in execution of key management functions (Denial of Service)
- Misuse of keys

Key management covers a broad number of services as defined above. However, it is not necessary for the Trusted Mobile Platform to standardize all of those areas. Specifically, areas that do not affect interoperability SHALL not require standardized protocols. These areas of key management that do not require standardized protocols still require a trusted platform to define the security requirements that must be met in order to mitigate the threats outlined above. These requirements are defined by **[SWAD]** and **[HWAD]**, and include the key management areas:

Key Generation	*Key generation* involves the creation of key material locally on a platform
Key Storage	*Key storage* involves the temporary, short-term placement of key material and related attributes when the key is not in use by a platform. In contrast, key archiving is the long-term storage of key material and related attributes on storage outside of the platform.
Key Installation	K*ey installation* is the setting up of a key for legitimate use on a platform (e.g. loading from key storage).
Key Usage	*Key usage* involves the use of key material by elements of a platform.
Key Deletion	*Key deletion* involves the removal of key material and related attributes from a particular platform.
Key Recovery	*Key recovery* is the process of restoring a previously archived (and possibly deleted) key to a platform

Key destruction involves deleting all archival copies of key material, and key revocation involves an authority revoking a key's association with an identity. These actions are beyond the scope of the Trusted Mobile Platform specifications, since it requires services beyond the client platform, and involve the external key infrastructure.

Key Destruction	*Key destruction* includes key deletion plus the destruction of all key archives stored off the platform for the key being destroyed.
Key Revocation	*Key revocation* is removal *of* a key from a registry, and optionally publishing information that the key association is no longer valid

Therefore, the Trusted Mobile Platform Protocol specification defines the key management services of registration, certification, and distribution of keys.

Key registration	● *Key registration* includes insertion of a key into a public or private registry service which associates that key with an entity – e.g. a trusted platform. ● *Key registration* and *key certification* may be performed in a single transaction. (See 7.3.4 - *Key Registration and Certification*.)
Key certification	*Key certification* includes the use of one or more authorities to create a certificate and certify a key, (usually pertaining to a public component of an

	asymmetric key pair.)
Key distribution	*Key distribution* involves the transport of key material and related attributes from one platform to another. This includes the possible negotiation of key attributes and parameters with a remote entity. (See .7.3.6 - *Key Distribution.*)

7.3.2 Key Types

The Trusted Mobile Device maintains several key types. These include TCG specified keys, and Trusted Platform specified keys. Some of these keys are protected by the TCG/TPM, while others exist in a protected key ring (*e.g. protected under a key from the SRK*).

The TCG specific keys include: the Endorsement Key (EK), Attestation Identity Keys (AIK), and the Storage Root Key (SRK). Additional storage keys may be created by the user, system or applications, and a TPM maintenance key may be specified by the TPM manufacturer.

There are a set of TCG credentials that exist as an X.509 certificate, that operate as root keys for endorsement agents. These include the Endorsement Credential, Conformance Credentials, the Platform Credential, and Validation Credentials. There are one or more Trusted Third Party root certificates that sign the AIK certificates.

The Trusted Mobile Platform specification defines a set of keys to supplement the TCG specified keys. These include a User Identity Key (UIK) for use in authenticating an end-user, Platform Encryption Keys (PEK) for use in platform protocols where a user specific key is not necessary or recommended, and temporary keys that are used to protect sessions. , A Trusted Mobile Device MAY have a set of Code Signing Keys and Maintenance Keys. The former as TTP keys representing entities that are trusted to sign downloadable code for the platform and the latter for protecting platform updates and patches by the OEM or manufacturer.

The diagram below shows the recommended breakdown of keys and how they should be protected in a Trusted Mobile Device.

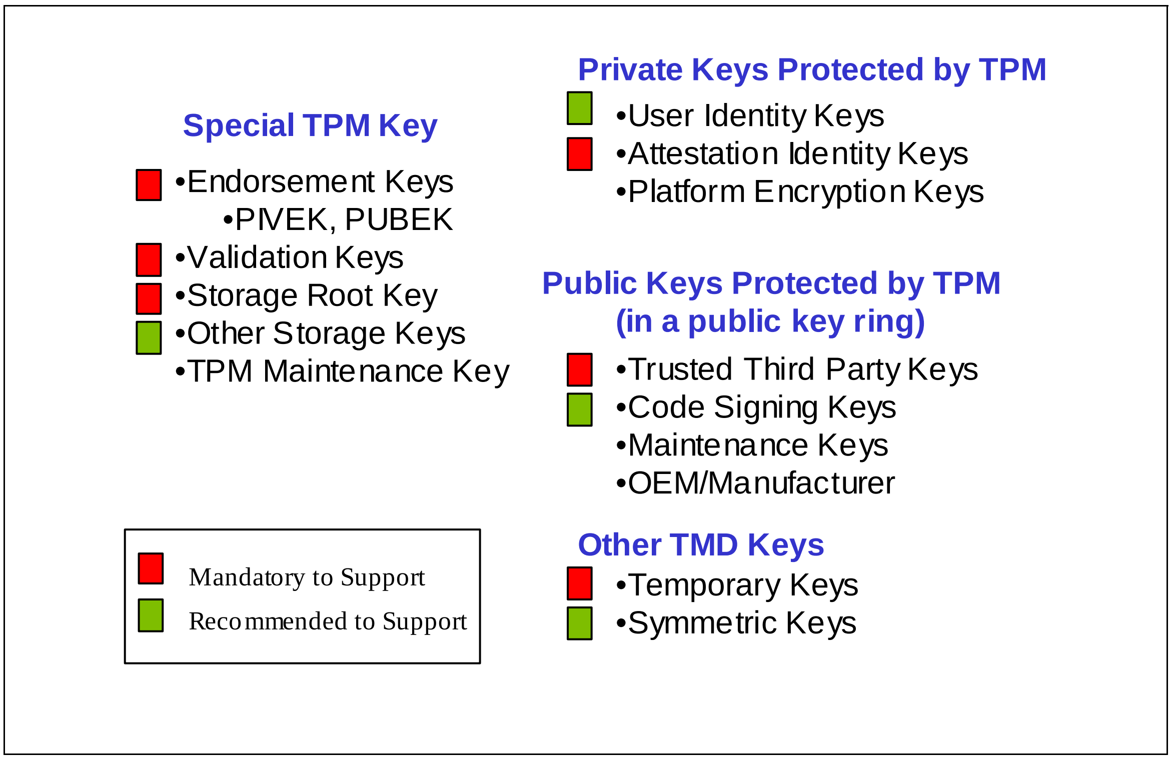


Figure 7-2 Trusted Mobile Device Key Types

7.3.3 Key Registration and Certification

Protocols for key registration and certification are described in this document in Section 5 Attestation and Supporting Infrastructure. These sections cover the creation, registration, and certification of Attestation Identity Keys (AIKs).

Other keys on the platform that require registration and certification include:

Trusted Code Signing Keys	Used to sign trusted boot code, trusted applications, and code updates.
User Identity Keys	Used to identify and authenticate a platform user to the platform, to the network, or network entity for the purpose of granting access rights and privileges.

These keys MAY use registration and certification mechanisms defined in **[PKIX].**

7.3.4 Key Management Infrastructure

Trusted Mobile Devices depend on a Public Key infrastructure for various key management services. The specification of these infrastructure components is beyond the scope of this specification. It is recommended that Trusted Mobile Devices comply with **[PKIX]** for interfacing with a PKI.

7.3.5 Key Distribution

Key distribution is an important part of several protocols in the Trusted Mobile Platform. Methods for key distribution are well known and varied. Trusted Mobile Platform proposes two types of key distribution protocols for symmetric keys (see section 7.3.5.1 - *Symmetric Key Distribution Protocol*). The distribution of asymmetric keys may be provided through **[PKIX]**.

7.3. 5.1 Symmetric Key Distribution Protocols

This specification defines three mechanisms for symmetric key distribution. The first uses Diffie-Hellman to produce a new, shared secret, and the second uses asymmetric keys to share key material that is used to create symmetric keys. This protocol is fully described in Section 4.7 (*Context Establishment*). In order for a high degree of trust to be associated with the creation of keys, these functions must be rooted in the TPM of the trusted platform. The second one is public key-based key exchange and described in the following subsection. The third one is to use an already-shared symmetric key to exchange a new key. This protocol is described in Section 8.3.4 (*Two-Way Authentication*).

7.3.5.1.1 Public Key-Based Symmetric Key Distribution Protocol

The following protocol is based on the symmetric key distribution scheme provided by SSL and TLS and provides a mechanism for two parties to authenticate one another (through the signatures) and share key data. This scheme is preferred since it can be accomplished by the cryptographic functions provided in current TPMs. The key created with these key distribution schemes can be used with symmetric encryption algorithms, message authentication codes, of HMAC.

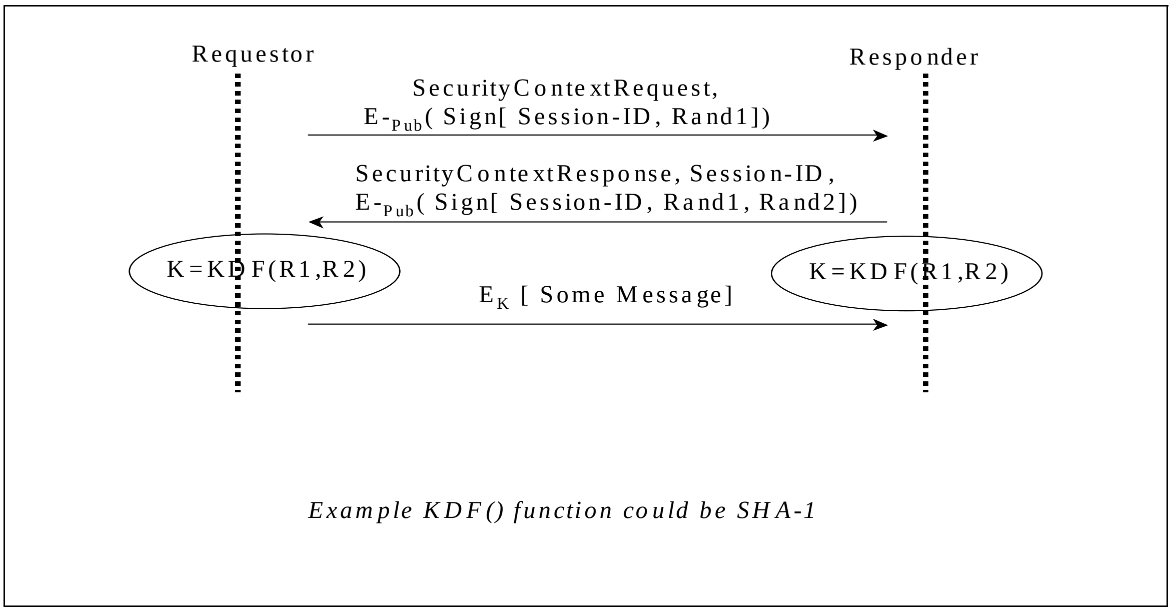

Requestor
Responder
SecurityContextRequest,
E-Pub(Sign[Session-ID, Rand1])
SecurityContextResponse, Session-ID,
E-Pub(Sign[Session-ID, Rand1, Rand2])
K=KDF(R1,R2)
K=KDF(R1,R2)
EK [Some Message]
Example KDF() function could be SHA-1

Figure 7-3 Public Key-Based Symmetric Key Distribution

This abstract authentication scheme will be bound to a specific wire format. This is intended to include binding to WS-Security as mentioned in Section 4.7 (*WS-Security*) in future revisions of this specification.

7.3.5.2 Use of SSL and TLS

The use of SSL and/or TLS is recommended by this specification in order to create cryptographically protected tunnels between the communicating parties. Such a practice will enhance communication security by providing basic confidentiality and integrity of the messages as the messages pass between end-points. The additional levels of security protection (e.g. SOAP message protections) are provided to ensure protection beyond the communication channel, as well as provide stronger security services (e.g. non-repudiation proof of source, trusted platform attestation, etc...).

Although the basic use of SSL and TLS are valuable, additional trust can be achieved by extending the services within these protocols. Particularly, tying the exchanges to attestation values from the TPM can leverage trust measurements into the authentication phase of the TLS exchanges.

[TLSExt] specifies a series of extensions to TLS that allows custom extensions to be defined for the client-server exchanges. The format of this extension is show below:

```
struct {
     ProtocolVersion client_version;
     Random random; SessionID session_id;
     CipherSuite cipher_suites<2..2^16-1>;
     CompressionMethod compression_methods<1..2^8-1>;
     Extension client_hello_extension_list<0..2^16-1>;
} ClientHello;

struct {
     ExtensionType extension_type;
     opaque extension_data<0..2^16-1>;
} Extension;
```

Trusted Mobile Platform recommends an extension to TLS using a *Platform_Config_Flag* as an extension that identifies the use of a *Platform_Config_PDU*. The *Platform_Config_PDU* sent from the server contains the selection of PCR registers or a nonce value. The client returns a *Platform_Config_PDU* with the signed PCRs or nonce. The certificate containing the AIK public key is provided through the Client Certificate message.

7.3.5.3 Related Standards

ISO11770-1 “Information Technology – Security Techniques – Key Management, Part 1: Key Management Framework” **[ISO11770-1]**

ISO7498-2 “Information Processing Systems – Open Systems Interconnection, Basic Reference Model, Security Architecture” **[ISO7498-2]**

NISTIR 4972 “A Study of OSI Key Management” **[NISTIR 4972]**

8 Access Control Architecture

8.1 Introduction

8.1.1 Scope

The scope of this section is to provide access control architecture for the Trusted Mobile Platform project. The objective of the section is to recommend a security solution for implementing mCommerce-enabled Trusted Mobile devices. The guiding design principles for the access control architecture proposed in this section are:

- Guarding against the threats described in the WG2 and WG3 Threat Model documents [WG2ThreatModel][WG3ThreatModel].
- Making full-use of the TPM security capabilities to enhance the platform and transaction security.
- Meeting the security requirements described in the Phase Zero document **[PhaseZero].**
- Enabling the Trusted Mobile Platform protocols.
- Making full use Java's Runtime Security capabilities, while extending them to take advantage of the TPM.

8.1.2 Overview of Access Control Mechanisms

The mobile phone is rapidly evolving into much more than a wireless telephone. It is transforming into a personal trusted device, with the ability to handle a wide variety of new services and applications such as banking, payments, ticketing and secure access-based operations.

The decision of which access controls to implement is based on organizational policy and on two generally accepted standards of practice: separation of duties and least privilege. For controls to be accepted and, therefore, used effectively, they MUST not disrupt the usual workflow; more than it should be necessary (may place a burden onto administrators, auditors, or authorized users). To ensure that access controls adequately protect all of the resources, it is necessary to first categorize the resources.

Policies establish levels of sensitivity (e.g., confidential) for data and other resources. These levels should be used for guidance on the proper procedures for handling objects — for example, instructions not to copy. They may be used as a basis for access control decisions as well. In this case, users (or owners) are granted access to only those resources at or below a specific level of sensitivity. Labels may be used to indicate the sensitivity level of electronically stored objects. For the Trusted Mobile Platform, the sensitivity levels need to be defined and named based on the types of secrets that must be protected (e.g., cryptographic keys) for the specific implementation, and not necessarily following commonly used naming schemes (e.g., Confidential) used in other frameworks such the Multi-Level Security (MLS).

In addition, the access control policy may be based on compartmentalization of resources. For example, access controls may all relate to a particular project or to a particular field of endeavor. Implementation of the access controls may involve either single compartments or combinations of them. These units of involvement are called categories, though the term “compartment” and “category” are often used interchangeably. Neither term applies to restrictions on handling of data. Individuals may need authorization to all categories associated with a resource to be entitled access to it or to any one of the categories.

The access control policy may distinguish among types of access as well. For example, billing applications may be authorized to read credit files, but modification of such files may be restricted to those responsible for compiling credit data.

One advantage of the use of sensitivity levels is that it allows security measures, which can be expensive, to be used selectively. Although the use of sensitivity levels may be costly, it affords protection that is otherwise unavailable and may well be cost-justified in many organizations.

Policy-based controls may be characterized as either mandatory or discretionary. With *Mandatory Access Controls* (MACs), only administrators and non-owners of resources may make decisions that bear on or derive from this policy. Only an administrator may change the category of a resource, and no one may grant a right of access that is explicitly forbidden in the access control policy. MAC (as indicated in *Table 8-2 Access Control Matrix*) is only required for certain types of secrets (e.g., SRK and AIK keys) and private data (e.g., User Profiles), which necessitate tightly-guarded access. Discretionary Access Control (DAC) is used for all other objects.

Access controls that are not based on the MAC policies are characterized as *Discretionary Access Controls* (DACs) or as “need-to-know” controls. The latter term connotes least privilege — those who may read an item of data are precisely those whose tasks entail the need.

It is important to note that mandatory controls are prohibitive (i.e., all that is not expressly permitted is forbidden), not only permissive. Only within that context do discretionary controls operate, prohibiting still more access with the same exclusionary principle.

Access controls can extend beyond limiting which subjects can gain what type of access to which objects. Administrators can limit access to certain times of day or days of the week. Typically, the period during which access would be permitted is 9 a.m. to 5 p.m. Monday through Friday. Such a limitation is designed to ensure that access takes place only when permitted, to discourage unauthorized use of data. Further, subjects’ rights to access might be suspended when needed.

When subjects leave an organization altogether, their rights MUST be terminated rather than merely suspended.

8.2 Terminology

Table 8-1 Terminology

Term	**Definition**
Access Control	A process by which use of system resources is regulated according to security policies and is permitted by only authorized entities according to those policies.
Access Control List (ACL)	A representation of who (user, group) can perform particular actions or Resources.

Term	Definition
Attribute	A descriptor that represents a characteristic of an object or subject.
Authentication	Authentication is the process of ascertaining the validity of either the device or the server’s identity.
Authorization	The process of allowing or denying a principal access to a resource, based on an Access Control Matrix (or a corresponding ACL), membership to a group, role attribute, rule, or some other evaluation criteria.
Confidentiality	Confidentiality is the ability to keep contents secret from all but the two entities exchanging a message. It does not limit the visibility of the message (being able to eavesdrop), but it does prevent the interpretation of the data being transmitted. Effectively this prevents the contents of a message being understood by anybody but the intended sender and intended recipient.
Credentials	Credentials are elements that are required to prove authenticity (e.g. username and password).
Discretionary Access Control (DAC)	An access control service that enforces a security policy based on the identity of system entities and their authorizations to access system resources. This service is termed "discretionary" because an entity might have access rights that permit the entity to enable another entity to access some resource.
Integrity	Integrity is the ability for a message or a system to maintain its content or at a minimum, have the ability to detect modification or corruption of its content.
Mandatory Access Control (MAC)	An access control service that restricts access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity. This kind of access control is called "mandatory" because an entity that can access a resource may not enable another entity to access that resource.
Object	A system element that contains or receives information.
Permission	The authority-based right that allows a user to perform an action.
Principal	A system entity whose identity can be authenticated.
Resource	The object of value that a user or system is attempting to access.
Role	An attribute that represents a collection of permissions.

Term	Definition
Session	A continuous connection between the device and the server established for the purpose of carrying out one or more operations.
Security Domain	An environment or context that is defined by security models and a security architecture, including a set of resources and set of system entities that are authorized to access the resources. One or more security domains may reside in a single administrative domain.
Security Policy	A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.
Subject	A system entity that causes information to flow among objects or changes the system state.
User	The person in possession of a personal trusted device and able to verify himself/herself to that device.

8.3 Architecture

8.3.1 Control Domains

Figure 8-1 illustrates the 6 Control Domains defined for the Trusted Mobile Platform architecture. The Owner Domain is the Root Domain, which owns all of its resources and which initiates any transaction. The Owner Domain interacts with all other 5 domains based on defined policies. The trust relationships between the 6 domains are defined by an Access Control Matrix (see *Table 8-2 An Example Access Control Matrix*) and are manifested in Policies. Policies are instantiations of the access control permissions can be found in the Access Control Matrix.

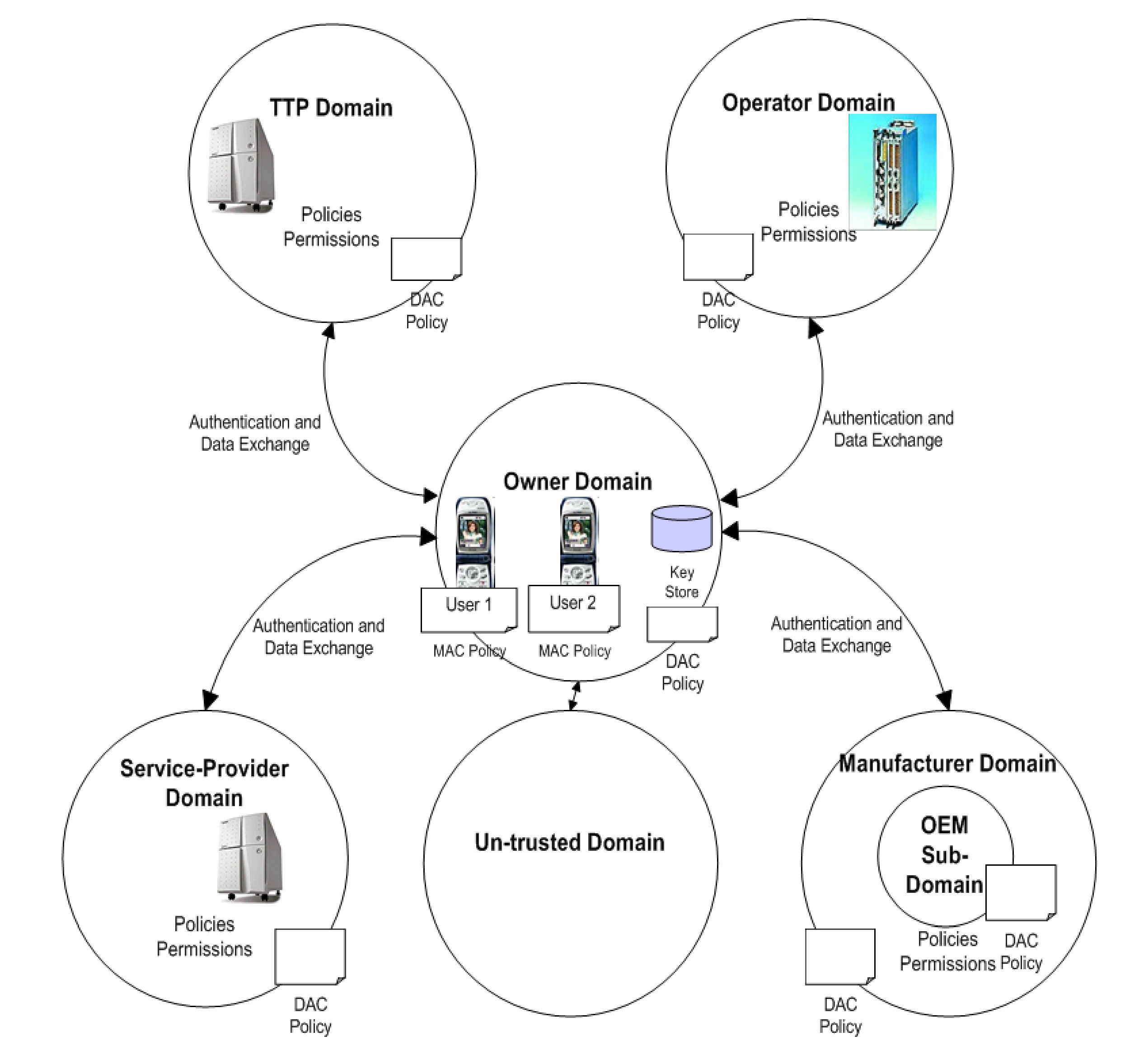


Figure 8-1 Trusted Mobile Platform Control Domains

The proposed access control architecture is based on a combination of Mandatory Access Control (MAC) and Discretionary Access Control (DAC) mechanisms. Various research results have shown that MAC is essential for the security of the whole system and that MAC mechanisms are very effective in supporting complex relationships between different entities in the computing environment, such as in the case of the Trusted Mobile Platform, where various secrets and trust elements need to be protected to ensure the trustworthiness of the platform **[Loscocco 1998]**. In MAC, the administrator imposes access control policies and object owners cannot change that policy. DAC policies, however, can be utilized for all other non-sensitive information, as manifested in *Table 8-2 Access Control Matrix*.

MACs can simply be implemented based on security labels associated with each object and each user. A label on an object is called a security classification. The resulting MAC policies MUST be transparent to the user.

8.3.2 Notes of CLDC, MIDP, MExE, and OSGi

This section is informative.

The purpose of this section is to shed some light on the limitations of the most popular Java-based models for mobile device security. Although the Trusted Mobile Platform access control uses some of the main concepts of the models described in this section (e.g., control domains), great care is taken in avoiding the limitations of the CLDC **[CLDC],** MIDP **[J2ME],** MExE **[3GPP]** and OSGi **[OSGi]** models. The goal is to learn limitations from those examples and define a powerful access control architecture for the Trusted Mobile Platform.

Currently, as the JVM and API specifications for mobile devices, Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP) are standardized and the reference implementations based on those specifications have been shipped **[J2ME]**.

However, as for the security of Java applications, only a byte-code verifier and a sandbox model are defined on CLDC. The verifier is expected to verify that the byte-code of the Java applications that conform to the Java language and the virtual machine specification. In the sandbox model, all Java programs are treated as potentially suspicious. Therefore, the sandbox model restricts all Java applications to use pre-defined safe functions within the mobile device. This creates a great constraint on using the Java applications on mobile devices. For example, Java applications cannot access private user data or hardware resources on the device due to safeguards against malicious applications from affecting the TMD. If these applications are considered "trusted" , the application has the ability to access those functions. Java applications that are considered "trusted" can provide more functionality on mobile devices.

In MExE **[3GPP]**, a security domain is introduced to define permission of application actions in the device including access control to resources and information in the device. Three security domains, which are Operator Domain, Manufacturer Domain and Third Party Domain, and available functions (actions) for each domain are defined. Applications not belonging to the above domains are regarded as Un-trusted and their functions are largely restricted. Though detailed actions of Java APIs in the terminal are not defined in the MExE specification, the API actions should be defined in a consistent way with policy of the security domain.

There are several problems in implementing an access control framework in the MExE specification described above. Certificate configuration management must be supported to handle an expired certificate. If the number of available domain is large, the verification time becomes unacceptably long because the device must verify all of the public keys corresponding to the domain for each program certificate. A way to give a certificate and signature to a program is undefined in the MExE specification.

In addition to the problems in validating and determining a domain of the program, there are problems relevant to downloading a Java application in CLDC and MIDP specifications. An application descriptor file (ADF) is used in downloading a Java application into the device. The ADF contains information about the Java application to be downloaded, such as a URL for the JAR file of the downloaded application, a size of the JAR file.

In order to maintain validation of the ADF, a certificate and signature are considered to be attached to the ADF. This constitutes a problem, as the ADF size will be slightly larger as the size for the certificate and signature to be added to the similar JAR file, as described above. Additionally, if a signature is attached to the JAR, malicious alternation of the JAR can be detected, but not that of the ADF. The MExE specification does not provide a solution to the alternation problem (See *Section 7.2* for more on *Core Software Download*).

The Core Platform Expert Group of the Open Services Gateway Initiative (OSGi) consortium **[OSGi]** needed to be able to change the permissions associated with code running at various service gateways. This change needed to be specified remotely and without having to reload the code at any of the service gateways. Due to footprint constraints (the deployment platform must be something small like Java 2 Micro Edition (J2ME), they were not able to leverage the dynamic permissions design implemented in J2SE version 1.4 **[J2SE].**

The Java 2 security model is too restrictive for an environment such as the OSGi service platform where multiple services from different vendors may be downloaded on demand, without a corresponding entry in the system policy file. Additionally, the permissions associated with a service's bundle of downloaded code may need to change over time, for example, when the bundle gets updated or when its configuration changes.

The OSGi specification defines a *`PermissionAdmin`* service that provides a standard interface for assigning Java 2 permissions to bundles based on their location. This allows for permissions to be setup before a bundle is installed on an OSGi service platform, and for the permissions to be modified at any time during the bundle's lifecycle. Updates to the permissions become effective immediately and persist across restarts of the OSGi service platform.

The scope of the OSGi specification is focused on defining a services gateway; the OSGi specification does not purport to be a sophisticated component model for developing complex applications. As such, the limitations of OSGi are not shortcomings per se, but result from the fact that some developers may try to build complex applications with it.

8.3.3 Access Control Matrix

The Access Control Matrix is a useful model for understanding the behavior and properties of access control systems. This matrix defines the trust relationships between the control domains and sub-domains. The implementation of the access control matrix can be based on a combination of Access Control Lists (ACLs), permission files, and an enforcement engine, such as Java's Security Manager and Access Controller. Policies are based on the relationships defined in the Access Control Matrix.

Four types of accesses are necessary to define the relationships between the objects and subjects. The first type, File Access, which includes the following permissions: Read (**R**), Write (**W**), and Execute (**E**). The second type is Message Access, which is necessary because of the need to control the exchange of messages between trusted and non-trusted domains and subjects. Message Access includes the following permissions: Send (**S**) and Receive (**RC**). The third type of access is Process Access, which controls the start and termination of processes such as Core Software Download, including the following permissions: Initiate (**I**), and Terminate (**T**). The fourth type of access is Key Accesses. This access type includes Create (**C**) and Use (**U**)

The following table defines an example of the access control matrix for a Trusted Mobile Platform and contains the subjects, objects, and the types of access allowed. Since the subjects also interact with each other, and hence the need to establish trust relationships between them, they are also considered as objects in this architecture (see *Table 8-2 An Example Access Control Matrix*).

The access control matrix reference (see *Table 8-2 An Example Access Control Matrix*), describes the type of access a subject can have in an object for any subject-object pair. These types of access are either mandatory (MAC-based) or discretionary (DAC-based). Note that Table 8-2 is only an example for the e-ticket scenario, and not a part of normative specification. However, an implementation of the Trusted Mobile Platform should define a similar access control matrix to clearly define the mandatory access control policy.

Table 8-2 An Example Access Control Matrix

Objects / Subjects	SRK	AIK	Other keys in TPM	eTicket	Location Data	User Profiles	Audit Log	MAC Policy	DAC Policy	Core Software Download	Owner	User’	Operator	Manufacture	Non-trusted user’	TTP’	Service Provider’
Owner (O), incl. system software	C_M, U_M	C_M, U_M	C_D, U_D	X	R_D, W_D	R_D, W_D	R_M, W_M	R_M, W_M	R_D, W_D	I_D, T_D	NA	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D
User (U)	U_M	U_M,	C_M, U_M	R_M, (manifest only)	R_M,	R_M, W_M	R_M	R_M	R_D, W_D	I_M, T_M	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D
Operator (OP)	U_D	U_D	C_D, U_D	X	R_M, W_D	R_D	R_M,	R_M	R_D, W_D	I_D, T_D	S_D, RC_D	S_D, RC_D	NA	X	S_D, RC_D	S_D, RC_D	S_D, RC_D
Manufacturer (M)	U_D	U_D	X	X	X	X	R_M	W_M R_M	R_D, W_D	I_D, T_D	S_D, RC_D	S_D, RC_D	X	NA	X	S_D, RC_D	X
Non-trusted User (N)	X	X	X	X	X	X	X	X	X	X	S_D, RC_D	S_D, RC_D	S_D, RC_D	X	X	X	S_D,
TTP	U_D	U_D	C_D, U_D	X	R_D	R_D	R_D	X	X	X	S_D, RC_D	S_D, RC_D	S_D, RC_D	S_D, RC_D	X	S_D, RC_D	S_D, RC_D
Service Provider (S)	U_D	U_D	C_D, U_D	R_D, W_D	R_D	R_D	X	X	R_D, W_D		S_D, RC_D	S_D, RC_D	S_D, RC_D	X	RC_D	S_D, RC_D	S_D, RC_D

Legend for Access Types:

Key Access	*C*	*Create*
	U	*Use*
File Access	*R*	*Read*
	W	*Write*
Process Access	*I*	*Initiate*
	T	*Terminate*
Message Access	*S*	*Send*
	RC	*Receive*
$()_M$		*Mandatory Access Type*
$()_D$		*Discretionary Access Type*
X		*No Access*
Example	R_M	*Read Mandatory Access Type*

This access control matrix represents the access relationships for the Trusted Mobile Device and SHOULD be the basis for its standard policy. In this table, mandatory policies are expected to be enforced on every Trusted Mobile Device of the same model. On the other hand, discretionary policies can be set depending on the device models (set by the manufacturer), the configuration of the device (set by the owner or the operator), or in some cases the user's preference (set by the user). Implementations may take different approaches to realize the policies. For example, some policies are expected to be built-in hardware features (such as protection of TPM keys). Other policies could be hard-coded in the system software or enforced by lack of access features. Yet, other policies could be implemented using more generic access control mechanisms as described in the Software Architecture Description **[SWAD]**.

The second column of the table (SRK) shows the access control relationships for the Storage Root Key. The device owner (or the device's system software that represents the owner's privilege) must be able to create a SRK. This is a part of the mandatory policy and marked as C_M in the table (mandatory creatable). On the other hand, the user must be able to use this key for any operation that requires the use of a TPM. This is marked as U_M in the table (mandatory usable). The use of the SRK is necessary in general for other domains, but the DAC policy may limit the use of the SRK for certain domains. Therefore, for other domains such as OP and TTP, they are marked as U_D (discretionary usable). Non-trusted users are not allowed to use the TPM capabilities, so the use of SRK is marked as X (no access) in the table.

Attestation Identity Keys (AIKs) are also created by the device owner. The use of an AIK is subject to the discretionary policy. For other keys in the TPM, the user should always be able to create and use such keys. For other domains, the use would be determined by the discretionary policy.

E-ticket is a special type of data object for Trusted Mobile Devices. It is created by a ticket issuer (a service provider) and redeemed by a ticket redeemer (another service provider). Because an e-ticket needs special handling, only the user can access the manifest part of the ticket. Section 5.7 provides additional details on how the Remote Data Binding Protocol protects e-tickets.

Location data and user profiles are the user's private data and therefore must to be properly protected. Additionally, the user should always be able to see and modify (in the case of user profiles) the information. In some case, the location data is provided by the system software and in other cases the network operator determines the location data. Therefore both the owner (system software) and the network operator are marked as discretionary write (W_D) for the location data.

The audit log should be written by the system software. It may contain sensitive private data so access to the audit logs needs to be limited. On the other hand, the device owner and the manufacturer must be allowed to access the log for problem resolutions purposes.

The MAC policy should be determined by the combination of the manufacturer and the owner. On the other hand, the user and the network operator can participate in the determination of the DAC policy. In some cases, some service providers MAY also need to set a DAC policy for its own data (such as an e-Ticket).

Downloading and updating the core software also require access control. The owner, the user, the operator, or the manufacturer should be able to initiate and terminate software update. However, there is a strong requirement that the end user must always be able to override the download operation. For that reason, the user has mandatory initiation and mandatory termination rights in the table.

8.3.4 Two-Way Authentication

Two-Way (Mutual) authentication is quite simple conceptually. The client and server MUST prove their respective identities to each other before performing any application functions. The central principal of two-way authentication is that neither party MUST "trust" the other before identity has been proven. What this means in practical terms is that the service MUST be able to determine who the client is *without asking the client* and the client MUST be able to determine who the server is *without asking the server*.

The value of a server being able to authenticate a client is well known. The value of a client being able to authenticate a server is less understood. Authenticating a server enables the client to trust the information it gets from the server and to feel secure in sending sensitive information to the server. The ability of a client to authenticate a server is particularly important in mobile applications that support delegation of the client's security context (in other words, the client authorizes the server to act as its delegate in accessing additional servers, such as a bank or a credit bureau, or network resources).

In a Trusted Mobile Device, two-way authentication can be implemented using one of several ways. SSL/TLS, which is recommended for the Trusted Mobile Device, has a mutual authentication capability. Also, Section 4.7 describes session establishment protocols that authenticate both parties. While the session establishment protocols can be used during initialization, the scheme proposed in this section extends the authentication process to future two-way authentication sessions.

In the rest of this subsection, we propose a simple two-way authentication scheme, which assumes that the user is already registered with the operator. This abstract authentication scheme will be bound to specific wire formats. It is intended that a binding to WS-Security is included in Section 4.7 in future revisions of this specification.

At initialization, the following operations SHALL be performed before any mutual authentication can take place:

- Collect device integrity metrics.

- User re-logs into the operator’s (or service provider’s) network.

- The operator (or service provider) checks the mobile device’s integrity. The mobile device will be trusted only if the reported PCR value is trusted.

The following diagram describes the proposed two-way authentication mechanism, which assumes a secret (HMAC) key is shared between the trusted user and the operator (or TTP).

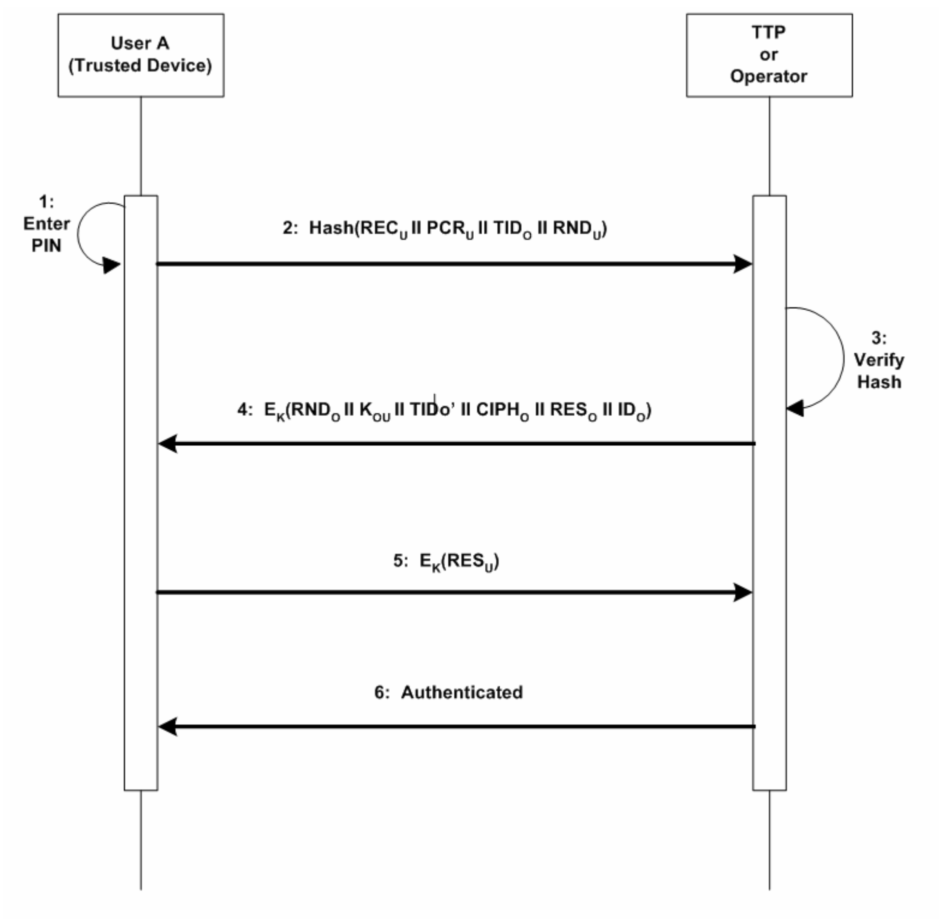

User A (Trusted Device)
TTP or Operator
1: Enter PIN
2: Hash(REC_U II PCR_U II TID_O II RND_U)
3: Verify Hash
4: E_K(RND_O II K_OU II TIDo' II CIPH_O II RES_O II ID_O)
5: E_K(RES_U)
6: Authenticated

Figure 8-2 Sequence Diagram for Two-Way Authentication

The following table summarizes the notations used in the above sequence diagram.

Table 8-3 Notation for Two-Way Authentication

Symbol	Definition
||	Concatenation operations.
AU	User authentication algorithm.

CIPH	A string of bits used to conceal the new temporary identity TIDO’ whilst it is in transit between the operator and the user.
EK	Shared-key encryption.
KOU	Key input
PCRU	Platform Configuration Register value that binds the message to the specific trusted platform in use.
RECU	A user-specific identity (such as the Home Location Register [3GPP] or a subset of it), which helps authenticate the sender and guarantees entity-authentication.
RESO, RESU	Challenge-responses generated by the operator and the user, respectively.
RNDU, RNDO	Random challenges generated by the user and the operator, respectively.
TIDO	A (temporary) identity is used to identify the trusted user to the operator with which she is currently registered. It is known both to the user and to the current operator.
TIDO’	The new temporary user identity for use with the TTP. TIDO’ replaces the current temporary identity TIDO.

A description of the sequence diagram follows. The User of a Trusted Mobile device MUST first enter a unique PIN number (or password/pass-phrase) to identify herself. In **step 1**, the hash of the concatenation of four separate values is sent to the TTP. The four values are: REC_U, PCR_U, TID_O, and RND_U. The hash value is easier to transport and verify than four different values.

Note that RND_U MUST be generated using the trusted hardware’s RNG services. PCR_U binds the message to the specific trusted platform in use.

Then, in **step 3**, the TTP verifies the hash value. If the verification passes, an encrypted message that includes the concatenation of (RND_O II K_{OU} II TIDo’ II $CIPH_O$ II RES_O II ID_O) is sent to the trusted user, **step 4**. The already-shared encryption E_K is used to encrypt the messages in order to protect the authentication message while in transit.

Then, RES_O is calculated using the user authentication algorithm A_U with key input K_{OU} and data string input the concatenation of RND_O, RND_U and TID_O’. The user authentication algorithm A_U takes input a secret key and a data string and outputs a check value RES_U. RES_U is calculated using the user authentication A_U with the key input K_{OU} and data string input the concatenation of RND_U and RND_O.

Please note that CIPH MUST be calculated using an identity hiding algorithm C_U with secret key input K_{OU} and data string input RND_U and that the identity hiding algorithm C_U MUST take as input a secret key and a data string and then output a string CIPH used to conceal a user identity.

In **step 5**, the encrypted value RES_U of is sent to the TTP. And finally, in **step 6**, an authentication message is sent to the trusted user.

Note: This section discusses authentication between two network entities (e.g., a TMD and a server). Authentication of a user against a TMD, for example by a PIN code or biometrics, is discussed in Section 10 of **[SWAD]**.

8.3.5 Policies

A security policy is a set of rules regarding the use of security services (e.g., mutual authentication between servers is required) and the details of security service implementation (e.g., a key length of at least 128 bits is required).

It MUST be assured that code, data, processes, and system resources are protected by access control policies. Access control policies determine what entities can create, read, modify, execute, and change access to resources. The code that enforces the policies, the rules that define the policies and the linkage between them MUST be protected.

Policies SHALL be defined and used within the Trusted Mobile Platform framework to control interactions within the framework. Policies SHALL be used to govern aspects of the system as a whole, examples including policies related to security, privacy and message handling.

Privacy policies SHALL be used to proactively define how and when personal information may be released, to whom, for what purpose, and, for how long. Such policies will have implications on the selection and use of security services. For example, appropriate confidentiality services are required to maintain the privacy of personal information during transit or storage. As a result, mechanisms designed to ensure confidentiality MAY be used to reduce the risks of inappropriate information disclosure. Such privacy policies may also have an impact on what information is routed.

Non-repudiation requirements SHALL introduce policies for logging and audit record maintenance, the use of timestamps and digital signatures, and more stringent authentication.

Section 8.5 below provides more details on the authorization, policy language and enforcement mechanisms.

8.4 An Example Access Control Implementation

This section is informative.

The generic architecture of Access Control of Trusted Mobile Device is described in the Software Architecture Description **[SWAD]**. This subsection illustrates one example implementation of access control. In this implementation, the security features and policies that are supported by the mobile device are enabled by a component called Common Access Control Service (CACS) Layer, which extends Java's `SecurityManager` and `AccessController` classes. At runtime, the CACS layer decides what policies are to be enforced when a connection request is made (both for inbound and outbound connections). Based on the domain (see *Auditing Capabilities of Trusted Mobile Platform*), device, device type and whether the domain or device is trusted or un-trusted, the CACS layer can enforce application level authentication, encryption of the session and any other specific access policies.

The CACS layer (see *Figure 8-3 Common Access Control Services (CACS) Architecture on Trusted Mobile Device*), needs information regarding devices and domains as well as services before it can take a decision whether or not to allow access and if so, to what services. This information is manifested in the Access Control Matrix (see *Section 8.3.3*) and is described in the Hash of Policy files, which are securely stored on the TPM's Data Integrity Registers (DIRs).

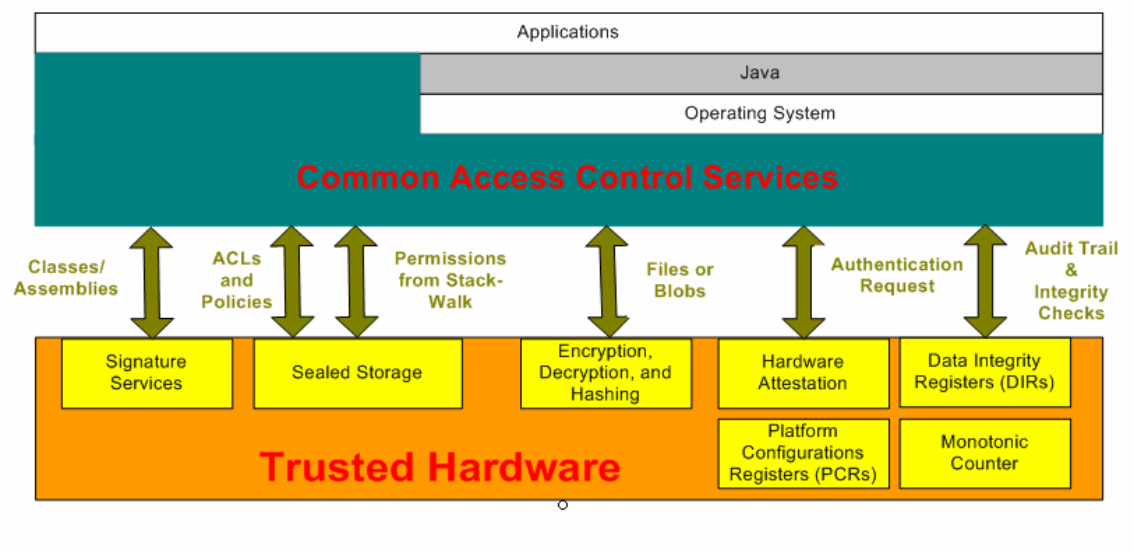

Applications
Java
Operating System
Common Access Control Services
Classes/ Assemblies
ACLs and Policies
Permissions from Stack-Walk
Files or Blobs
Authentication Request
Audit Trail & Integrity Checks
Signature Services
Sealed Storage
Encryption, Decryption, and Hashing
Hardware Attestation
Data Integrity Registers (DIRs)
Platform Configurations Registers (PCRs)
Monotonic Counter
Trusted Hardware

Figure 8-3 Common Access Control Services (CACS) Architecture on Trusted Mobile Device

The proposed CACS layer modularizes the security services and abstracts their enforcement as an extension to the Java runtime environment. This is possible because, in Java 1.2, the content of a security policy is totally separated from not only the implementation mechanism but also the interfaces. This leaves maximum room for extensibility and modularization. It also allows the policy files to be configured entirely separately from the runtime environment, thus reducing the complexity of system administration. Also, in Java 1.2, the access control algorithm is cleanly separated from the semantics of the permissions that it is checking. This allows the reuse of the access controller with application-specific permission classes that are introduced later (after device shipped).

Mutual authentication is also implemented at the CACS layer using Java's extensible `LoginContext` classes. In the proposed architecture, applications remain independent of the underlying authentication services (e.g., TPM, SIM card, biometric). (*See Figure 8-4 CACS-based authentication.)*

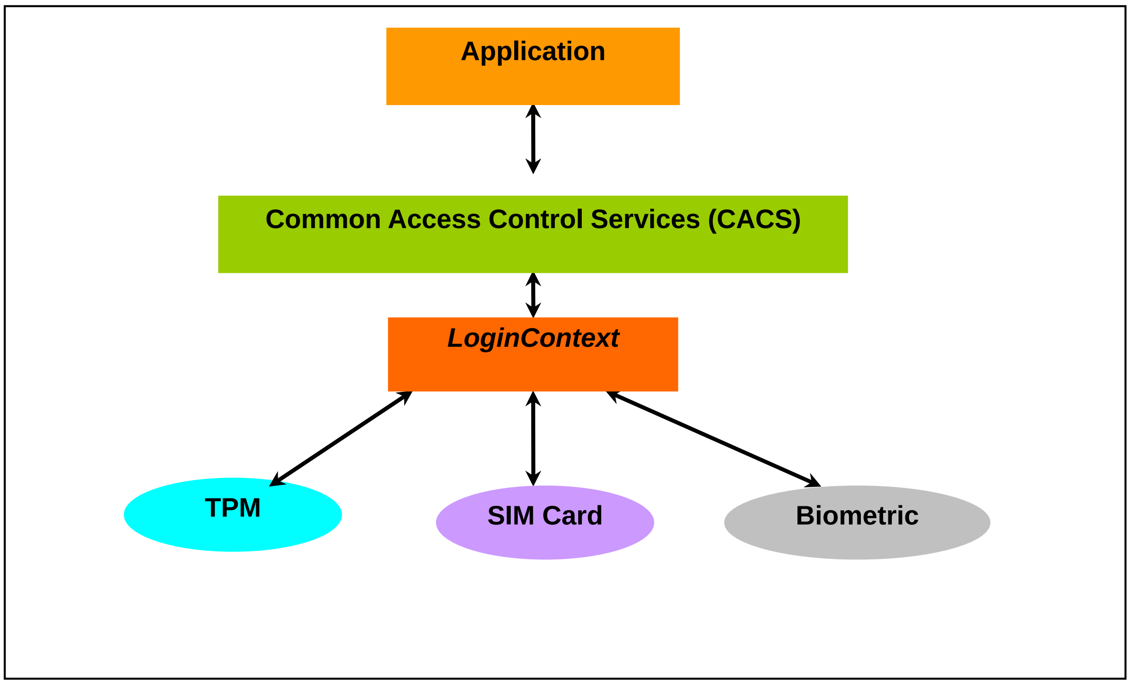


Figure 8-4 CACS-based authentication

The CACS layer provides security services based on the following capabilities of the TPM:

- The TPM strengthens the mutual authentication process by generating the random challenge RND_U (and possibly RND_O).
- The TPM stores event logs in its registers for auditing purposes.
- The TPM will provide cryptographic services, to include asymmetric key generation, digital signatures, hashing, and encryption.
- The key used during access control can be securely stored using the TPM's Secure Key Storage capabilities.
- The runtime auditing functions will make full use of the TPM's PCR registers, Secure Data Storage, and Session Storage capabilities.

8.5 Authorization, Policy Language and Enforcement

8.5.1 Introduction

The development of an Access Control system requires the definition of the subjects and objects against which authorizations must be specified and access control must be enforced. Therefore, there is a need to define a common authorization model for all the trust domains.

There is also a need for a common language for expressing security policies. If implemented across the Trusted Mobile Platform's trust domains, a common policy language allows the domain users and owners to manage the enforcement of all the elements of its security policy in all of its components. Managing security policy MAY include some or all of the following steps: writing, reviewing, testing, approving, issuing, combining, analyzing, modifying, withdrawing, retrieving and enforcing policy.

8.5.2 Authorization

A good authorization model (and supporting language) should support authorizations at all levels of granularity, including individual documents and elements within them. The object granularity for which authorizations should be specified should span from single elements to attributes within individual documents, where elements and attributes can be referenced by means of path expressions.

SAML **[SAML]** and XACML **[XACML]** SHOULD be used to specify and express policies and authorizations. WS-Security **[WS-Security]** SHOULD be used for managing and exchanging trust relationships. Authorizations SHOULD be made accessible to each access control domain. Each access control domain that receives such a request will lookup authorization policies that are relevant to the authorization request. It then appends these to the request, digitally signs the modified request, and forwards it to the next access control domain in the ordered list of the ACL.

Authorizations MUST be stored in a safe and protected place if they are not digitally signed, so that tampering with them is prevented.

Authorizations MUST be either *positive* (permissions) or *negative* (denials). The reason for having both positive and negative authorizations is to provide a simple and effective way to specify authorizations applicable to sets of subjects/objects with support for exceptions.

8.5.3 Policy Language and Enforcement

The major requirements for a Policy Language to support the Trusted Mobile Platform access control architecture, which includes 6 separate Control Domains (as well as MAC- and DAC-based policies), are:

- The ability to independently administer multiple policies controlling access to the same resources.

- The ability to select algorithms for reconciling conflicting policies.

- The ability to efficiently locate all the policies that are potentially applicable to a given decision without sacrificing the flexibility described above.

XACML **[XACML]** meets the three requirements above by including two related languages: a policy language that defines access control and a request/response language in which queries and decisions are conveyed. The policy language is used to express access control policies (who can do what on what). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses).

While SAML **[SAML]** assertion provides a mechanism for making authentication and authorization assertions, a vocabulary is also needed for expressing the rules needed to make authorization decisions. One XML vocabulary created specifically for expressing authorization rules is the XACML. XACML defines: an XML vocabulary for expressing authorization rules, an XML vocabulary for expressing a variety of conditions to be used in creating rules, how rules are to be combined and evaluated, and means for creating policy statements, a collection of rules applicable to a subject.

While the XACML specification uses the SAML definitions for subjects and actions, it itdefines rules as *targets*, *effects* and *conditions* (where a target includes resources, subjects and actions), and *effects* (where an effect is either "Allow" or "Deny").

A deployed XACML access control system SHOULD work the following way:

- A subject S seeks access to some object O.
- The subject submits its query to the entity protecting the resource. This entity is called a Policy Enforcement Point (PEP).
- The PEP forms a request (using the XACML request language) based on the attributes of the subject, action, resource, and other relevant information.
- The PEP then sends this request to a Policy Decision Point (PDP), which examines the request, retrieves policies (written in the XACML policy language) that are applicable to this request, and determines whether access should be granted according to the XACML rules for evaluating policies.
- That answer (expressed in the XACML response language) is returned to the PEP, which can then allow or deny access to the requester.

Each XACML policy document SHOULD have exactly one Policy or `PolicySet` root XML tag. A `PolicySet` is a container that can hold other Policies or `PolicySets`, as well as references to policies found in remote locations. A Policy represents a single access-control policy, expressed through a set of `Rules`.

XACML defines and describes "layering" between XML entities to clearly distinguish between security technologies that create policy, collect the data required for policy evaluation, evaluate policy, and enforce policy.

Because a generic *Policy* or `PolicySet` may contain multiple policies or *Rules*, each of which may evaluate to different access control decisions, XACML needs some way of reconciling the decisions each makes. In XACML, this is done through a collection of *Combining Algorithms*. Each algorithm represents a different way of combining multiple decisions into a single decision. XACML utilizes *Policy Combining Algorithms* (used by `PolicySet`) and *Rule Combining Algorithms* (used by *Policy*). The *Deny Overrides Algorithm* is an example of these indicating that no matter what, if any evaluation returns *Deny*, or no evaluation permits, the final result is also `Deny`. These *Combining Algorithms* MAY be used to build up complex policies to suit the specific Trusted Mobile Platform implementation needs.

It is RECOMMENDED that the *Policy Enforcement* work in the following fashion:

- The PDP receives a request to determine if access should be granted to a resource based on rule sets, or policies, that are already defined.
- Access control information is kept in a physically separate repository that is referenced when a request is made.
- `XPaths` are defined within tags in the XML resource that inform the parser to check the XACML policies and where to find them.

8.5.4 XACML Policy Example

This section is informative.

Assume that a corporation named e-Ticket Corporation (e.g., `eticketcorp.com`) has an access control policy, which states: any user with a name in the "eticketcorp.com" namespace is allowed to perform any action on any resource.

In XACML, this policy should be expressed as follows. An XACML policy consists of header information, an optional text *description* of the policy, a *target*, one or more *rules*, and an optional set of *obligations*.

The header for this Policy is:

```
<?xml version=1.0" encoding="UTF-8"?>
<Policy
   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
   xmlns:function="urn:oasis:names:tc:xacml:1.0:function"
   xmlns:identifier="urn:oasis:names:tc:xacml:1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy
      http://www.oasis-open.org/.../cs-xacml-schema-policy-01.xsd"
   PolicyId="identifier:example:eTicketPolicy1"
 RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides
">
```

The above header introduces the XACML Policy, and then it defines a set of namespace prefixes. Element names used in the document should be treated as if they were prefixed by the URN *`urn:oasis:names:tc:xacml:1.0:policy`*. Also, element names prefixed with the words *`function`, `identifier`, or `xsi`* should be treated as if they were prefixed instead by the specified URNs. Then, a URL to the schema for XACML policies is provided.

The name *`eTicketPolicy1`* is assigned to this policy instance. The name of a policy should be unique for a given PDP so that there is no ambiguity if one policy is referenced from another policy.

Finally, the algorithm that will be used to resolve the results of the various rules that may be in the policy is provided. The *deny-overrides* algorithm specified here says that, if any rule returns a *Deny Access* result, the policy must return a *Deny Access* result. If all rules return a *Permit Access* result, then the policy must return a *Permit Access* result. The algorithm also says what to do if errors were encountered in evaluating any rules, and what to do with rules that do not apply to a particular request.

Now, let's provide a text description of the policy. This description is optional.

```
  <Description>
    e-Ticket Corporation access control policy
  </Description>
```

Hereinafter, the decision requests to which this policy applies will be described. If the subject, resource and action in a decision request do not match the values specified in the *Target*, then the remainder of the policy does not need to be evaluated. This *Target* section is very useful for creating an index to a set of policies. In this (e-Ticket) implementation example, the *Target* section says the policy is applicable to any decision request.

```
<Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
```

The <*Rule*> element specifies a rule in the policy. Just as for a policy, each rule must have a unique identifier for any PDP that will be using the policy. The identifier for this rule is then specified.

As an *Effect*, this rule will generate a `Permit` result, if the rule evaluates to TRUE. Rules may have an *effect* of either `Permit` or *`Deny`*. In this case, the rule will return a result of `Permit`, meaning that, as far as this one rule is concerned, the requested access should be permitted. If a rule evaluates to FALSE, then it returns a result of *`NotApplicable`*. If there is some error in evaluating the rule, the rule returns a result of *`Indeterminate`*. As mentioned above, the rule combining algorithm for the policy tells how various rule results are combined into a single policy result.

```
<Rule
     RuleId="identifier:example: eTicketPolicy1"
     Effect="Permit">
```

An optional <*Description*> element specifies description of the rule.

```
<Description>
      Any subject with a name in the eticketcorp.com domain
      can perform any action on any resource.
</Description>
```

The *<Target>* element specifies the target of the rule. The *Target* of a rule describes the decision requests to which this rule applies. If the subject, resource, and action in a decision request do not match the values specified in the rule Target, then the remainder of the rule does not need to be evaluated, and a result of *NotApplicable* is returned.

```
    <Target>
```

This Target is similar to the Target of the policy itself, but with one important difference. In this case, a specific value is spelled out and the subject in the decision request must match this value. The <*SubjectMatch*> element specifies a matching function (*MatchId*), a pointer to a specific subject attribute in the request (*SubjectAttributeDesignator*), and a literal value. The matching function will be used to compare the value of the subject attribute with the literal value. Only if the match returns TRUE will this rule apply to a particular decision request. If the match returns FALSE, then this rule will return a result of *NotApplicable*.

```
    <Subjects>
       <Subject>
         <SubjectMatch MatchId="function:rfc822name-match">
           <SubjectAttributeDesignator
              AttributeId="identifier:subject:subject-id"
              DataType="identifier:datatype:rfc822name"/>
           <AttributeValue
            DataType="identifier:datatype:rfc822name">eticketcorp.com
           </AttributeValue>
         </SubjectMatch>
       </Subject>
     </Subjects>
     <Resources>
       <AnyResource/>
     </Resources>
     <Actions>
       <AnyAction/>
     </Actions>
    </Target>
```

Finally, the rule and the policy need to be closed. In this rule, all the *work* is done in the Target specification. In more complex rules, the Target may have been followed by a Condition (which could also be a set of conditions to be *AND*-ed or *OR*-ed together).

As mentioned above, this policy has only one rule, but more complex policies MAY have any number of rules.

```
  </Rule>
 </xacml:Policy>
```

Next example shows a decision request that may be submitted to a PDP using the policy above. In English, the access request that generates the decision request may be stated as follows:

John Doe, with a name *"johndoe",* wants to receive an e-Ticket from his Service Provider `eticketcorp.com` (see *Figure 8-2 Sequence Diagram for Two-Way Authentication* for other access types).

In XACML, the information related to this access request is formatted into a *Request Context* statement that looks as follows.

First, the header for the Request must be defined.

```
<?xml version="1.0" encoding="UTF-8"?>
 <Request
   xmlns="urn:oasis:names:tc:xacml:1.0:context"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
http://www.oasis-open.org/tc/xacml/1.0/sc-xacml-schema-context-01.xsd"
>
```

Then, the <*`Subject`*> element contains one or more attributes of the entity making the access request. There can be multiple subjects, and each subject can have multiple attributes. In this case, there is only one subject, and the subject has only one attribute: the subject's identity, expressed as a name, is *`johndoe`*.

```
  <Subject>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="identifier:rfc822name">
        <AttributeValue>johndoe</AttributeValue>
    </Attribute>
  </Subject>
```

The <*`Resource`*> element contains one or more attributes of the resource to which the subject (or subjects) has requested access. There must be only one <*`Resource`*> per decision request. There is only one attribute for the resource to which John Doe has requested access: the resource identity, which is *`http://eticketcorp.com/etickets/issued/johndoe`*.

```
   <Resource>
     <Attribute
        AttributeId="identifier:resource:resource-uri"
        DataType="xs:anyURI">
       <AttributeValue>http://eticketcorp.com/etickets/issued
/johndoe</AttributeValue>
     </Attribute>
   </Resource>
```

The *<Action>* element should contain one or more attributes of an action that the subject (or subjects) wishes to take on the resource. There can be only one action per decision request. The following code describes the identity of the action John Doe wishes to take, which is *Receive*.

```
  <Action>
     <Attribute
        AttributeId="identifier:action:action-id"
        DataType="xs:string">
       <AttributeValue>Receive</AttributeValue>
     </Attribute>
   </Action>
```

Then, close the request.

```
</Request>
```

A more complex request may contain some attributes not associated with any of the *subject*, the *resource*, or the *action*. These would have been placed in an optional *<Environment>* element following the action.

The *PDP* processing this request *context* locates the *policy* in its policy repository. It compares the *subject*, *resource* and *action* in the request *context* with the *subjects*, *resources* and *actions* in the *policy target*. Since the *policy target* matches the *<AnySubject/>, <AnyResource/>* and *<AnyAction/>* elements, the *policy* matches this *context*.

The *PDP* now compares the *subject*, *resource* and *action* in the request *context* with the *target* of the one *rule* in this *policy*. The requested *resource* matches the *<AnyResource/>* element and the requested *action* matches the *<AnyAction/>* element, but the requesting subject-id *attribute* does not match "*@eticketcorp.com".

As a result, there is no *rule* in this *policy* that returns a *"Permit"* result for this request. The *rule-combining algorithm* for the *policy* specifies that, in this case, a result of *"Deny"* should be returned. The response *context* looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
 <Response
    xmlns="urn:oasis:names:tc:xacml:1.0:context"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
http://www.oasis-open.org/tc/xacml/1.0/sc-xacml-schema-context-01.xsd"
>
```

The <*Result*> element contains the result of evaluating the decision request against the policy. In this case, the result is *Deny*. A policy can return *Permit*, *Deny*, *NotApplicable*, or *Indeterminate*.

```
  <Result>
    <Decision>Deny</Decision>
  </Result>
```

Finally, the response is closed.

```
</Response>
```

9 Related Standards

There are a number of security protocol standards. A Trusted Mobile Device is recommended to implement some of them while other standards may be optionally implemented. Yet another category is that the TMD can support the protocol using its unique capability such as the protected storage of the TPM or the attestation signature. This section describes the summary of these standards that are relevant to the Trusted Mobile Platform protocols. The following table shows the categorization of these standards.

Category Label	Description
Mandatory	This standard is an essential part of the Trusted Mobile Platform and MUST be implemented in all compliant TMDs.
Recommended	This standard constitutes a significant value-add to the TMD and therefore is RECOMMENDED to be implemented.
Optional	Implementation of this standard is OPTIONAL. If it is implemented, the other Trusted Mobile Platform protocols SHOULD take advantage of this protocol.
Supported	This standard can use the capabilities of TMD to enhance its security or functionality.
Transparent	Support of this standard is completely independent from the Trusted Mobile Platform capabilities.

Note: Details of how the standards marked as Mandatory, Recommended, or Optional are applied to the Trusted Mobile Platform protocols are given in the corresponding sections of this document.

9.1 TLS (Recommended)

The support for TLS (RFC-2246) **[TLS]** is RECOMMENDED. If a TMD supports TLS, it MUST have a TLS client features. It is RECOMMENDED that the TMD support TLS client authentication capability. It is RECOMMENDED that the device support the use of *Sign_RSA_PCR* described in Section 5.5.1. Also, a TMD SHOULD use the keys stored in the TPM whenever appropriate. A TMD MAY implement the server side TLS capability.

9.2 X.509 v3 (Recommended)

If a TMD supports the authentication based on public-key cryptography, X.509 v3 certificate MUST be supported. This specification does not specify any extensions to X.509 certificate attributes, but the other standards such as TCG **[TCG]** and PKIX **[PKIX]** requires their own specific attribute extensions.

9.3 TCG

TCG **[TCG]** defines a set of message formats regarding endorsement key credentials, platform credentials, identity key credentials, and validation data. Section 5 defines in detail how these messages are exchanged. A class 2 or 3 TMD must support **[TCG]** capabilities as defined in **[HWAD]**. A class 1 TMD device is RECOMMENDED to support software TPM or function similar to TPM as described in **[HWAD]**.

9.4 XML Security / WS-Security (Recommended, Profiled)

It is RECOMMENDED that profiled versions of XML Signature, XML Encryption, Exclusive Canonicalization, WS-Security, WS-Trust, WS-SecureConversation, SAML, and XACML are supported by the TMD. The details of the profiles are given in Section 4.

9.5 PKIX (Optional)

When the device has WS-Trust **[WS-Trust]** capability, the support for PKIX is OPTIONAL. However, if the device has no WS-Trust, it MUST implement PKIX, including certificate request, certificate retrieval, and certificate revocation.

9.6 SPKI (Supported)

Implementation of SPKI **[SPKI]** can use a signature key stored in the TPM. Also, TPM protected storage should be used for protecting keys.

9.7 IPSec (Supported)

Implementations of IPSec **[IPSec]** (RFC-2401) can use a signature key stored in the TPM. It also can use the protected storage for securely storing the symmetric secrets.

9.8 SSH (Supported)

Implementations of SSH can use a signature key stored in the TPM. It also can use the protected storage for securely storing the symmetric secrets.

9.9 PGP (Supported)

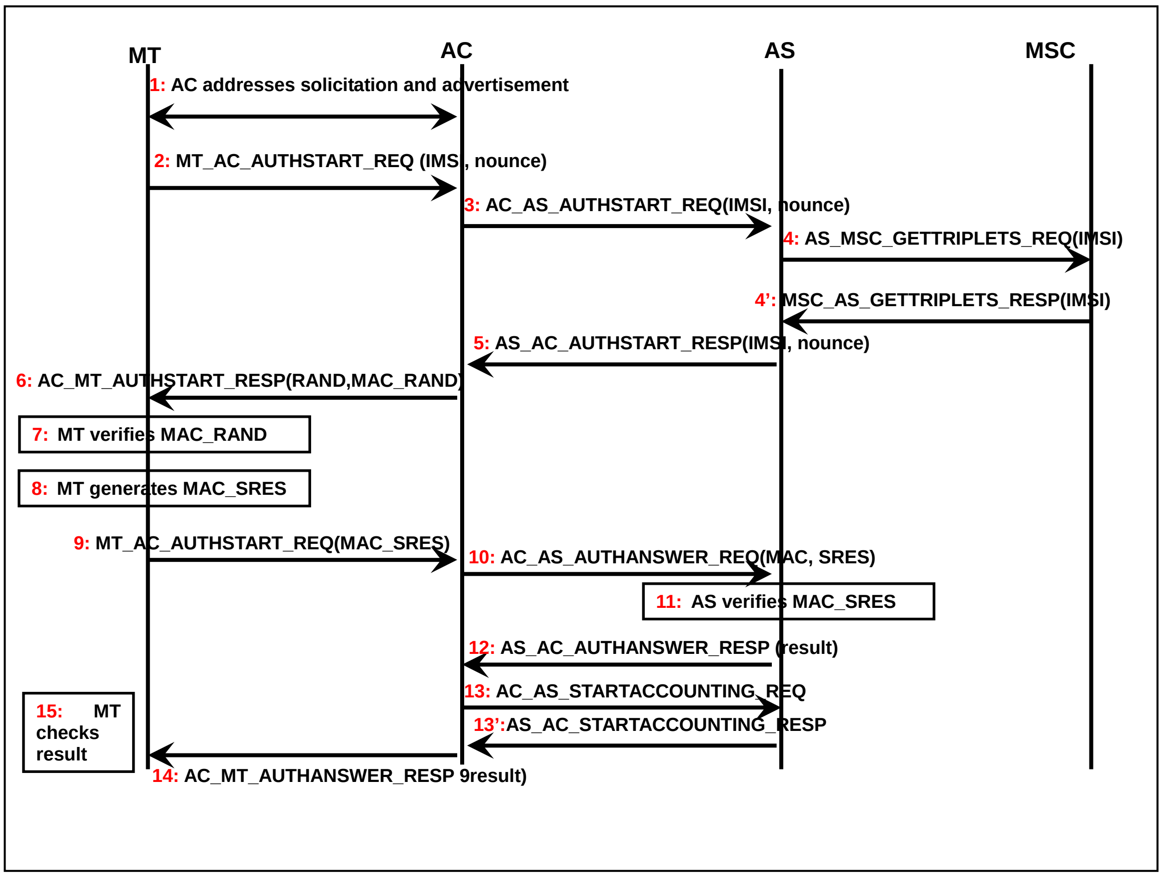
Implementations of PGP can use a signature key stored in the TPM. It also can use the protected storage for securely storing the symmetric secrets.

9.10 S/MIME (Supported)

Implementations of S/MIME can use a signature key stored in the TPM. It also can use the protected storage for securely storing the symmetric secrets.

9.11 GSM SIM-based Authentication (Transparent)

GSM defines its own authentication protocol based on the SIM card. This protocol is illustrated in Figure 9-1. Note that in this figure, the SIM card is depicted as MT, or Mobile Terminal. Since the SIM card directly talks to the AC (Access Control server), a TMD is unrelated to this protocol. Although the TMD should provide a means to relay these messages between the SIM card and the AC, this communication should be opaque to the TMD. In other words, the TMD is transparent to this protocol.


MT
AC
AS
MSC
1: AC addresses solicitation and advertisement
2: MT_AC_AUTHSTART_REQ (IMSI, nounce)
3: AC_AS_AUTHSTART_REQ(IMSI, nounce)
4: AS_MSC_GETTRIPLETS_REQ(IMSI)
4': MSC_AS_GETTRIPLETS_RESP(IMSI)
5: AS_AC_AUTHSTART_RESP(IMSI, nounce)
6: AC_MT_AUTHSTART_RESP(RAND,MAC_RAND)
7: MT verifies MAC_RAND
8: MT generates MAC_SRES
9: MT_AC_AUTHSTART_REQ(MAC_SRES)
10: AC_AS_AUTHANSWER_REQ(MAC, SRES)
11: AS verifies MAC_SRES
12: AS_AC_AUTHANSWER_RESP (result)
13: AC_AS_STARTACCOUNTING_REQ
13':AS_AC_STARTACCOUNTING_RESP
15: MT checks result
14: AC_MT_AUTHANSWER_RESP 9result)

MT	Mobile Terminal
AC	Access Controller
AS	Authentication Server
MSC	(Operator's) Mobile Switching Center
IMSI	GSM International Subscriber Identity
SRES	Signed Response (using algorithms stored on SIM card)

Figure 9-1 GSM Authentication Protocol

10 Security Considerations

10. 1 Introduction

The Trusted Mobile Platform's objective is to address the multi-level framework associated in establishing a Trusted Mobile Platform. The architecture framework documents are structured and divided into three different specifications. The protocol specification (*this document*) addresses several characteristics in the exchange and data flow between its associated trusted parties. The Hardware Architecture Description **[HWAD]** addresses high-level hardware architecture design of a trusted mobile device. The Software Architecture Description **[SWAD]** addresses key software components to develop a secure solution to maintain trust between hardware, software and protocol environments.

There are several security considerations that any implementer of the Trusted Mobile Platform should pay attentions. For example, the Trusted Mobile Platform does not prevent or mitigate any Denial of Service (DoS) attacks. As the majority of the protocols defined in the protocol specification relies on the trust infrastructure, the implementations need to understand the security levels of the trust infrastructure and should set appropriate policies. Policies need to be in place to determine on balancing the level of risks and the cost of implementing security mechanisms to mitigate the risks.

The implementer must keep in mind to address fundamental security policy considerations which are necessary for practical use (e.g., Proper SSL/TLS implementation/design and user privacy protecting examination). This section, at the time of this draft, has no answers to address against some of these key issues, for example interoperability and scalability. Hence, the scope of this document is left open ended for the Platform operators to apply other means of security (e.g., anti-virus, anti-spam, intrusion detection, and firewall). The specification does not address operational considerations.

10.2 Operational Issues after a Breach in Security

In addressing operational issues in an event of a security breach, the service operator MUST evaluate the breach of security. It is assumed that the security breach is executed through existing/new backdoor vulnerabilities on the Trusted Mobile Platform. The service operator MUST assess the vulnerability risk to the overall system. The system architecture of the trusted mobile platform is designed in mind to detect these anomalies; before vulnerability poses a threat onto the Trusted Platform. This section can only theorize what can transpose, facilitate and remedy the situation, such as factors which exist that may reflect onto the Platform; reliability of the device or the wireless network interference may prevent Trusted Platform to communicate and resolve vulnerabilities.

The possibility of exploiting backdoors and vulnerabilities may exist on the trusted mobile device. It is assumed that the network operators can fix the security vulnerability; by providing a security patch; however it is up to the user's will and has the option to accept the software update. These are one of many issues that need to be addressed, however unfortunately this specification can only contribute as guidelines and framework in keeping the network and the trusted mobile device secure.

These operational policies demonstrate different severity levels in assessing the security vulnerability risk to service operations; the service operator operational requirement would fit into the categories, as mentioned below:

10.2.1 Operational Policy One: Service Provider Continues Service without fixing vulnerability.

In policy one, the Trusted Mobile Platform continues to be in operation. The vulnerability may have the tendency to disrupt revenue-producing services (some of which may be providing real-time services, so any disruption will cause a significant loss) and thus; poses a security risk. The attack is either likely or unlikely, depending on the nature of the vulnerability and the likelihood of the service being targeted. The service provider may choose to continue communicating with the device by taking the risk. This may be justified by the nature of the service (whose value as being a real-time and/or mission critical service is very high) and the likelihood of the possible attacks.

10.2.2 Operational Policy Two: Service Providers will only allow clients that have installed the proper security patch.

In policy two, some mobile devices may have backdoors and the knowledge that vulnerabilities may exist. The trusted state of the mobile platform may be considered 'un-trusted' since the nature of the platform has been compromised. However, the service provider understands the ramifications of halting services. One applicable solution is to interrupt or limit the trusted services to mobile devices that pose a threat with backdoors. The service provider / mobile device manufacture may eventually provide a patch for this backdoor. As result users who do not apply the proper security patch, their trusted service may be hindered until they install the proper security patch. For users who choose not to install the proper security patch may quit the trusted service (due to limitation of trusted services, on the mobile devices that are effected), but the service provider will allow the ability to continue the trusted services to the rest of the customer base.

10.2.3 Operational Policy Three: Service Providers will only allow clients that have been 'reinitialized'.

In policy three, some mobile devices may have backdoors and the knowledge that vulnerabilities may exist. The trusted state of the mobile platform is considered 'un-trusted' since the nature of the platform has been compromised, in some form or fashion. The manufacturer reclaims/recalls the affected devices, reinitializes them, and redistributes them to the affected consumer base. This would be considered the most secure approach, but heavily impacts the business due to the service's unavailability to the affected consumer base; which could be prohibitively large. Under this situation, service provider may get frustrated in protecting its 'trusted state'. The service provider may then cancel and or stop the whole trusted service offering. Resulting in the attrition of the consumer base, service provider support and effecting overall business model.

11 Future Agenda

Many potential Trusted Mobile Platform protocols and/or protocol-bindings have been identified while drafting this specification, although some of them have not been fully addressed. This section describes some of such protocols that are possibly going to be defined in future versions of this specification. This section is informative. There is no intention to guarantee that protocols listed in this section will be defined as parts of any future versions of the Trusted Mobile Platform specifications.

11.1 Attestation and Supporting Infrastructure

Since **[TCG]** focuses on defining a set of structures, APIs, and credentials for the trusted platform subsystems, it requires additional network protocols to efficiently communicate with, and utilize the feature of the Trusted Mobile Platform from the networked entities. Some of such protocols are defined in section 5 to support functions such as creation of platform identities or validation of the platform integrity measurement. This section describes other potential protocols that are not defined in this specification.

11.1.1 Registration of Trusted Integrity Measurements

In order to attest to a particular state of a Trusted Mobile Device, the manufacturer (or its delegate) is responsible for keeping track of core software configurations and communicate with the validation entity to vouch for the integrity measurement of the trusted configurations. A network protocol for registering trusted configurations with validation entities may be defined.

11.1.2 Migration

[TCG] defines functions to securely archive migratable keys stored in the protected storage of the TPM to be transferred to other entities. A public key has to be authorized by the Trusted Mobile Device's owner before creating a migration archive, therefore only an authorized entity that has the associated private key can decrypt the archive. Network protocols for authorizing a public key, and creating and transferring the migration archive may be defined.

11.1.3 Maintenance

[TCG] defines functions to securely archive non-migratable keys stored in the protected storage of the TPM to be transferred to the manufacturer (or its delegate). Network protocols for creating and transferring the migration archive may be defined.

11.1.4 Backup of User Data

A backup of a user's confidential data on a Trusted Mobile Device can be securely performed using cryptographic protection provided by the TPM. For example, a copy of a user's data in the protected storage hierarchy of the TPM can be sent over the network without revealing its content, as long as the storage key protecting the data is not revealed. The backup of the storage key can be made secure using the migration or the maintenance feature of the TPM. (See Section *11.1.2 Migration* and *11.1.3 Maintenance*). The protocols for creating a backup of the data and restoring it over the network may be defined.

11.2 Access Control

Additional protocols for enhancing access control mechanisms discussed in section 8 may be defined. In particular, a remote entity may create an application domain and configure the Discretionary Access Control (DAC) policy of the domain. An example of such a use case is a service provider creating an application domain on the Trusted Mobile Device and configures the DAC of the domain, so the particular information used by the service can be updated only by the service provider. It is intended by authors to define a common AC language for Trusted Mobile Platforms in the future version of this specification.

Also, future versions of this specifications document, the access control section will further address other aspects including manageability, provisioning, and administration.

11.3 Key Exchange Protocol Bindings

A few key exchanging methods are defined in this document, but bindings to the network protocol are not defined. The protocol binding of the following key exchange methods may be defined.

Section 7.3.5.1.1	Public Key-Based Symmetric Key Distribution Protocol
Section 8.3.4	Symmetric Key-Based Key Exchange

See Section 4.7 for more information.

11.4 Device Management

This document defines requirements for device management (Section 7). Methods for the software download and installation may be defined in future versions of the Trusted Mobile Platform specification.

The protocols for the following device management capabilities described in Section 12 of **[SWAD]** are not defined in the current version of the Trusted Mobile Platform specifications. Future versions of the Trusted Mobile Platform specifications may define protocols for these capabilities.

- Configuration management
- Software and hardware inventory management
- Notification (alert)
- Logging
- Remote Monitoring
- Remote Diagnostics

12 References

12. 1 Normative References

[KEYWORDS]	S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997
[RFC2234]	"Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. November 1997. URL:http://www.ietf.org/rfc/rfc2234.txt
[CRMF]	RFC 2511, "Internet X. 509 Certificate Request Message Format", March 1999. http://www.ietf.org/rfc/rfc2511.txt.
[EXC-C14N]	"Canonical XML" Version 1.0, W3C Recommendation 15 March 2001. http://www.w3.org/TR/xml-c14n.
[HTTPBasicAuth]	RFC 2617, "HTTP Authentication: Basic and Digest Access Authentication", June 1999. http://www.ietf.org/rfc/rfc2617.txt.
[HWAD]	"Trusted Mobile Platform Hardware Architecture Description", version 1.0, 30 June 2003. http://www.trusted-mobile.org/.
[IPSec]	RFC 2401 "Security Architecture for the Internet Protocol", November 1998. http://www.ietf.org/rfc/rfc2411.txt
[ISO11770-1]	"Information Technology – Security Techniques – Key Management, Part 1: Key Management Framework"
[ISO7498-2]	"Information Processing Systems – Open Systems Interconnection, Basic Reference Model, Security Architecture"
[Minimal-WS-Security]	"SOAP Message Security: Minimalist Profile (MProf)" Working Draft 1.5, March 7, 2003. http://www.oasis-open.org
[NISTIR-4972]	"A Study of OSI Key Management"
[PhaseZero]	"Trusted Mobile Platform Phase-0 Document", version 1.0, 30 June 2003. http://www.trusted-mobile.org/
[SAML]	"Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1", May 2003. http://www.oasis-open.org/
[SOAP11]	W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/SOAP/
[SOAP12]	See **[SOAP12-Part0]**, **[SOAP12-Part1]** and **[SOAP12-Part3]**
[SOAP12-Part0]	W3C Recommendation 24 June 2003, SOAP Version 1.2 Part 0: Primer, SOAP Version 1.2 Part 0: Primer
[SOAP12-Part1]	W3C Recommendation 24 June 2003, SOAP Version 1.2 Part 1:

	Messaging Framework, http://www.w3.org/TR/soap12-part1/
[SOAP12-Part2]	W3C Recommendation 24 June 2003, SOAP Version 1.2 Part 2: Adjuncts, http://www.w3.org/TR/soap12-part2/
[SSL3]	A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996
[SWAD]	"Trusted Mobile Platform Software Architecture Description", version 1.0, 30 June 2003. http://www.trusted-mobile.org/.
[TCG]	Trusted Computing Group (TCG) Main Specification Version 1.1b", February 2002. http://www.trustedcomputinggroup.org/ (also previously known as Trusted Computing Platform Alliance (TCPA) Main Specification Version 1.1b).
[TCP/IP]	Defined in many RFC documents by the Internet Engineering Task Force. See http://www.ietf.org
[TLS]	RFC 2246, "The TLS Protocol" Version 1.0. http://www.ietf.org/rfc/rfc2246.txt.
[TLSExt]	RFC3546, "Transport Layer Security (TLS) Extensions", June 2003. http://www.ietf.org/rfc/rfc3546.txt.
[URI]	T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396.
[WS-SAML]	"Web Services Security: SAML Token Profile", Working Draft 06, 21 February 2003. http://www.oasis-open.org/.
[WS-Security]	"Web-Services Security: SOAP Message Security", Working Draft 12, Monday 21 April 2003. http://www.oasis-open.org.
[WS-Trust]	"Web Services Trust Language (WS-Trust)", Draft 18 December 2002, http://www-106.ibm.com/developerworks/library/ws-trust/
[WS-X509]	"Web Services Security: X509 Certificate Token Profile" Working Draft 05, 6th June 2003. http://www.oasis-open.org/.
[XACML]	"eXtensible Access Control Markup Language (XACML)" Version 1.0, OASIS Standard, 18 February 2003. http://www.oasis-open.org/
[XML-Encryption]	"XML Encryption Syntax and Processing", W3C Recommendation 10 December 2002. http://www.w3.org/Encryption/2001/
[XML-Namespace]	"Namespaces in XML", W3C Recommendation, 14 January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/.
[XML-Signature]	"XML-Signature Syntax and Processing", W3C Recommendation 12 February 2002. http://www.w3.org/Signature/.

12. 2 Informative References

[3GPP]	http://www.3gpp.org/
[Afaria]	http://www.xcellenet.com/public/products/afaria/afaria.asp
[Asokan1998]	Asokan, N., Shoup, V., Waidner, M., "Optimistic Fair Exchange of Digital Signatures", IEEE Journal on Selected Areas in Communications.
[Bluetooth]	Bluetooth SIG, "Specification of the Bluetooth System", Revision 1.1, February, 2001. http://www.bluetooth.org
[CLDC]	Java Community Process, "JSR-000139 Connected Limited Device Configuration 1.1 (Final Release)", http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html
[ISO14443]	International Standards Organization, ISO/IEC 14443, "Identification Cards – Contactless integrated circuit(s) cards – Proximity cards", April, 2000, http://www.iso.org
[J2ME]	http://java.sun.com/j2me/
[J2SE]	http://java.sun.com/j2se/
[Loscocco 1998]	P. Loscocco, et al. *"The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments".* Proceedings of the 21st National Information Systems Security Conference. October 1998.
[MIDP2.0]	"Mobile Information Device Profile for Java 2 Micro Edition", Version 2.0. http://java.sun.com/products/midp/
[OSGi]	http://www.osgi.org/
[PKIX]	RFC 2527, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", March 1999.
[PGP]	RFC 1991, "PGP Message Exchange Format", August 1996.
[ProvOTA]	C. Enrique Ortiz "Introduction to OTA Application Provisioning", November 2002. http://wireless.java.sun.com/midp/articles/ota/
[ProvMIDP]	"Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile", Version 1.0, May 7, 2001. http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf.
[S/MIME-MSG]	RFC 2663, "S/MIME Version 3 Message Specification", June 1999.
[SPKI]	RFC 2693, "SPKI Certificate Theory", September 1999.

13 Terminologies and Abbreviations

ACL	Access Control List
ADF	Application Descriptor File
AIK	Attestation Identity Key **[TCG]**
B2B	Business to Business
B2E	Business to Employee
CA	Certificate Authority
CACS	Common Access control Service
CE	Conformance Entity
CLDC	Connected Limited Device Configuration
CORBA	Common Object Request Broker Architecture
DAC	Discretionary Access Control
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DIR	Data Integrity Register. **[TCG]** requires at least one Data Integrity Register (DIR). A DIR holds 160-bit value and is in TCG-shielded location. The DIRs may be used for any purpose, but may be useful to applications taking measurements, but where an appropriate PCR cannot be identified.
EK	Endorsement Key **[TCG]**
eTicket	A downloaded object that asserts that a fare or admission has been paid. An eTicket may include additional attributes such as the time and the venue of an event the ticket may be used for admission.
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications, http://www.gsmworld.com/.
HLR	Home Location Register **[3GPP]**
HMAC	Hashed Message Authentication Code
HTTP	Hyper Text Transfer Protocol
HTML	Hyper Text Markup Language
IrDA	Infrared Data Association
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAR	Java Archive
MAC	Mandatory Access Control / Message Authentication Code
MExE	Mobile Station Application Execution Environment, http://www.mobilemexe.com/.
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extensions **[RFC2045]**
OSGi	Open Services Gateway Initiative, http://www.osgi.org/.
PCR	Platform Configurations Register **[TCG]**. A PCR consists of a 160-bit field that holds a discrete integrity measurement. The PCR data structure is in a TCG-shielded location. **[TCG]** requires a TPM implementation to provide

	16 or more independent PCRs. These PCRs are identified by index and are numbered from 0 (that is, PCR0 through PCR15 are required for TCG compliance).
PEK	Platform Encryption Key
POP3	Post Office Protocol Version 3 **[RFC1081]**
S/MIME	Secure MIME **[S/MIME-MSG]**
SAML	Security Assertion Markup Language **[SAML]**
SMTP	Single Mail Transfer Protocol **[RFC821]**
SOAP	Simple Object Access Protocol **[W3C]**
SRK	Storage Root Key **[TCG]**
SSH	Secure Shell
SSL	Secure Socket Layer **[SSL3]**
TCB	Trusted Computing Base **[TCG]**
TCG	Trusted Computing Group **[TCG]**
TLS	Transport Layer Security **[TLS]**
TMD	Trusted Mobile Device
TPM	Trusted Platform Module. A keywords starts with TPM_ refers to a command or structure defined in **[TCG].**
TSS	TCG Supported Software **[TCG]**. A keyword starts with TSS_ refers to a command or structure defined in **[TCG]**.
TTP	Trusted Third Party
UIK	User Identity Key
URI	Universal Resource Identifier **[URI]**
VE	Validation Entity **[TCG]**
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

Appendix A. Change History (Informative)

Revision	Date	Section	Description
Initial drop	02/13/2003		The initial version of this document.
0.01	03/03/2003	6	Merged: rev0_01-1__20030227-Section6-hm
0.02	03/17/2003	2,3,5,8	Merged: 03 Protocols Overview.doc Access Control Architecture V0_1- Aissi 02 TMP Protocol Design General Principles 05 TCPA Enablement_20030313_taiga
0.02-2	04/10/2003	4,5,7,8	Merged: 07 Management(edited by Takeshita, Ver0,1) Access Control Architecture V0_1- Aissi.doc 04 WS-Security Profiles 05 TCPA Enablement_20030401_sachiko-q Added author names of Section 9
0.03	06/02/2003	3, 5 Appendix C	Merged’ TCPA Enablement in section 5 with some parts moved to Appendix C (Taiga) Scenarios (Sachiko)
0.04	06/12/2003	3, 4,5,6	Restructured section 3. (by Dave W. [tmp-wg-two 95] and Sachiko) Update on section 4 by Hiroshi [tmp-wg-two 92] and Daniel [tmp-wg-two 94] Added schemata in section 5. Added schemata and description in section 6 by Hiroshi [tmp-wg-two 93]
0.05	06/13/2003	7,9,10,11	Update from NTT DoCoMo (TMP_ PROTOCOL_DRAFT.DCM.REVISION-ONE.doc) Added references and abbreviations.
0.06	06/14/2003	1	Update on Introduction [tmp-wg-two 91]
0.07	06/16/2003	8	Update on Access Control Architecture [tmp-wg-two 112]
0.08	06/25/2003	1, 2, 3, 4, 5, 6, 7, 9, 10,11, 12, 13	“ Introduction” update [tmp-wg-two 152]. “ Protocol Design General Principles” update [tmp-wg-two 155]. “ Protocols Overview” update [tmp-wg-two 167]. “ WS Security” update [tmp-wg-tw 149]. “ TCPA Enablement” update [tmp-wg-two 150] with new “Remote Data Binding” [tmp-wg-two 146] as Section 5.7. “Terminal Management” update [tmp-wg-two 166]. “ Key Management” [tmp-wg-two 145] inserted as Section 7.4. “ Fair Contract Signing Protocol” update [tmp-wg-two 151]. New “Related Standard” [tmp-wg-two 143] inserted as Section 9. “Security Consideration” update [tmp-wg-two 166]. New “Future Agenda” [tmp-wg-two 153] inserted as Section 11. Addition to section References and Definitions and Abbreviations.
0.09	06/25/2003	7.3, 8	“Core Software Download” example updated [tmp-wg-two 175] “Access Control” updated [tmp-wg-two 176]
0.10	07/01/2003	1, 4, 7, 8, 10, 11	“Introduction” update [tmp-wg-two 211] “WS-Security” update [tmp-wg-two 211] “Device Management” update [tmp-wg-two 214] “Access Control” update [tmp-wg-two 209] “Security Considerations” update [tmp-wg-two 215] “Future Agenda” update [tmp-wg-two 216]
0.11	07/04/2003		Final WG draft for internal reviews.
0.12	09/30/2003	all	Applied comments from joint WG2-3 discussions to resolve inconsistencies between documents.
0.13	10/07/2003	3.1 5.1 5.7 7.3.3	Too many TCG in a paragraph – sentences revisited. Too many TCG in paragraphs – sentences revisited. “TCG TPM” was changed to “TPM” Reference to section 5 title was corrected.

Revision	Date	Section	Description
		13.1	Sentences revisited.
0.90	10/15/2003	1	Removed "Over time, it is intended that the protocols specified in this document can be incorporated into appropriate protocol standards."
0.91	10/20/2003	all	Editorial changes from NTT DoCoMo. (filename: tmp_protocol_spec_rev0_90_20031015_DCM_Modified.doc)
0.92	10/20/2003	all	Changed domain names from tmpalliance.org or trustedmobile.org to trusted-mobile.org. Fixed wrong references and other editorial errors.
1.00	4/5/2004		Version 1.0 release.

Appendix B. TCG Technology for Attestation (Informative)

13.1 Creation and activation of TCG identity key

TCG defines a set of structures, APIs and credentials for creating and activating an Attestation Identity Key (AIK). This is one of the most important functionalities of trusted platforms and it is strongly related to the philosophy of the overall TCG design.

The description below is prepared for the better understanding of how TCG is designed to deal with the identity key and why the protocols are needed to support it. This section is for the informational purpose only and does not intend to override **[TCG]**.

13.1.1 Assumptions

1. Identity key is a “pseudonym” that is associated with the Trusted Platform Module (TPM). A platform with the TPM can create and use any number of identity keys. Note that each TPM already has its own unique key called an Endorsement Key, which is an asymmetric key-pair and whose private key never comes out of the TPM.

There are basically two reasons why an identity key is needed separately from the endorsement key:

- Security
- Privacy.

From security point of view, the endorsement key is the most critical long-term secret in TPM, thus it is better to avoid using it in cryptographic operations (encryption, signing) for utilizing TCG functionalities. Once the endorsement key for a certain TPM is revealed all security dependent on that TPM is compromised. From privacy point of view, using a single key in many different situations may lead to the concern of privacy breach. For example, if the endorsement key is visible to multiple services, they can correlate all access from the platform that has the TPM with this endorsement key.

2. There is a trusted third party that issues the credential that a certain identity key is come from a genuine TPM. This TTP is called Privacy CA. There can be multiple Privacy CAs, but each identity key is associated to exactly one Privacy CA when it is created.

Only the Privacy CA should be able to have the knowledge of the matching of the identity key and the endorsement key (i.e., the TPM) for the privacy reason. Most of the complication in the process of creating and activating an identity is derived to this requirement.

3. An identity key can only be used for the special ways of signing. It cannot be used for encryption or usual signing operation. More precisely, it can be used for two dedicated purposes: (1) to respond to integrity attestation, and (2) to certify that other keys are associated with the genuine TPM. The identity key can be used only via the special APIs (*`TPM_Quote`* and *`TPM_CertifyKey`*, respectively), and it always signs the determined TCG structure.

Nevertheless, it is possible to use the identity key for signing general data in special cases. *`TPM_Quote`* API accepts an external data as an input parameter and includes it in the data structure that is signed by the identity key. This data structure also contains the hash of integrity metrics, so the platform integrity at the time the data is signed can be proved by using the identity key.

13.1.2 Overview of the procedure

Figure 13-1 (*Creating and Activating an Identity Key*) illustrates the procedure of creating and activating an identity key.

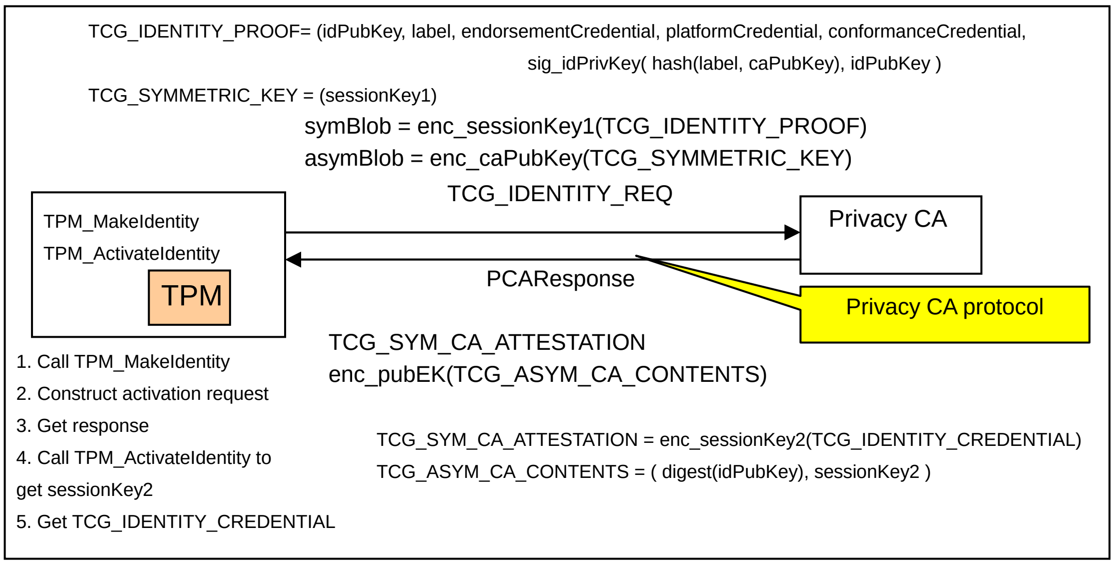

TCG_IDENTITY_PROOF= (idPubKey, label, endorsementCredential, platformCredential, conformanceCredential,
sig_idPrivKey(hash(label, caPubKey), idPubKey)
TCG_SYMMETRIC_KEY = (sessionKey1)
symBlob = enc_sessionKey1(TCG_IDENTITY_PROOF)
asymBlob = enc_caPubKey(TCG_SYMMETRIC_KEY)
TCG_IDENTITY_REQ
TPM_MakeIdentity
TPM_ActivateIdentity
TPM
Privacy CA
PCAResponse
Privacy CA protocol
1. Call TPM_MakeIdentity
2. Construct activation request
3. Get response
4. Call TPM_ActivateIdentity to get sessionKey2
5. Get TCG_IDENTITY_CREDENTIAL
TCG_SYM_CA_ATTESTATION
enc_pubEK(TCG_ASYM_CA_CONTENTS)
TCG_SYM_CA_ATTESTATION = enc_sessionKey2(TCG_IDENTITY_CREDENTIAL)
TCG_ASYM_CA_CONTENTS = (digest(idPubKey), sessionKey2)

Figure 13-1 Creating and Activating an Identity Key

The whole process can be divided into three independent processes:
1. Call the *TPM_MakeIdentity* function to create an identity and prepare for requesting an identity credential.

2. Send an identity request to Privacy CA and receive a response.

3. Call the *TPM_ActivateIdentity* function to activate an identity and obtain the identity credential.

** Step 2.) Involves the communication between the platform and the Privacy CA, and that is where the Privacy CA protocol is needed.

13.1. 3 MakeIdentity

When *TPM_MakeIdentity* is called, the TPM internally generates an identity key-pair and returns the public key of the identity key, signed as part the wrapped structure by its private key. The private key does not come out of the TPM in plaintext.

13.1.4 Identity certificate request and response

Once the key-pair is created, it can be used in the subsequent *TPM_Quote* and *TPM_CertifyKey* calls. However, it is useless at this stage because there is no proof that this key is associated with a genuine TPM.

To actually make use of the identity key, an identity certificate needs to be issued by the Privacy CA. Platform (or platform owner) constructs activation request and get a response.

TCG_IDENTITY_REQ (defined in Section 4.30.2 and 9.4.1 of **[TCG]**) is the structure sent as a message from the platform to the Privacy CA. It consists of the *TCG_IDENTITY_PROOF* structure (defined in Section 4.30.3 of **[TCG]**) encrypted by the session key, and the session key encrypted by the public key of the Private CA. The *TCG_IDENTITY_PROOF* structure includes a public key of the identity key to be certified, as well as a set of credentials for the platform.

PCAResponse (defined in 9.4.2 of **[TCG]**) is the structure sent as a message from the Privacy CA to the platform. It consists of the *TCG_SYM_CA_ATTESTATION* structure (Section 4.30.5 of **[TCG]**) encrypted by the second session key, and this session key encrypted by the public key of the endorsement key-pair.

As is seen, these request and response are very different from the one used for usual certificate request. This is designed so for the purpose of fulfilling the following requirements.

- For security reason, it should be avoided, whenever possible, to use the private key of the endorsement key-pair, because it is the most critical long-term secret in TCG system. Note that any part of identity request is not signed by the endorsement key.

- For privacy reason, the eavesdropper should not be able to know the mapping between the identity key and the endorsement key associated with the TPM.

Finally, some important details are omitted. For example, *TCG_IDENTITY_REQ* also includes a property called "label", which creates the cryptographic binding between *MakeIdentity* output and the request to the Privacy CA. Refer to **[TCG]** for the complete description of the protocol.

13.1.5 ActivateIdentity

TPM_ActivateIdentity (as defined in 9.3.4 of **[TCG]**) is an API that provides the functionality for decrypting the session key. Since the session key is encrypted by the endorsement key-pair, only the TPM that was originally used to generate the identity key can decrypt it. Once the decrypted *PCAResponse* session key obtains, the platform (or the platform owner) can recover the identity certificate.